TDA User's Manual

February 14, 2005

Götz Rohwer Goetz.Rohwer @ ruhr-uni-bochum.de
Ulrich Pötter Ulrich.Poetter @ ruhr-uni-bochum.de

Ruhr-Universität Bochum. Fakultät für Sozialwissenschaften, GB 1.
44780 Bochum, Germany.

Contents

1. Introduction

This introductory part of the TDA User's Manual contains the following sections.

1.1 Preface

1.2 Documentation explains the structure of this manual and our usage of cross-references.

1.3 Installation explains how to install the program on different platforms.

1.4 How to Use the Program provides basic information on invoking the program and using its commands.

1.5 The Example Archive explains how to find the examples that are part of the program's distribution.

1.6 Errors and Error Messages discusses errors that may occur when using the program.

1.7 Some General Commands describes a few commands that may be helpful in different contexts.

1.8 Command Flow Control describes commands that can be used to control command flow, e.g., `while` and `repeat`.

1.9 Programming Repetitions describes some commands that can be used to program repetitions, e.g., for bootstrap applications.

1.10 Using Macros explains how to define and use macros.

1.1 Preface

TDA, an abbreviation of *Transition Data Analysis*, is a computer program for statistical computations. Developing the program is a work in progress. First versions have been developed in 1989. Since then, several new features have been added. With version 5.7 (June 1994) the program has reached some state of maturity. This is reflected in the book by Blossfeld and Rohwer [1995] providing an introduction to techniques of event history modeling based on TDA. In fact, this book is supplemented by a disk containing, together with a lot of examples, version 5.7 of the program.

We then experimented with several different approaches to develop a more flexible and easier to use version of the program. In version 5.7 there was no clear distinction between commands, in the sense of basic procedures, and additional parameters to control the behavior of these commands. In the preliminary new versions, 6.02 and 6.03, we tried to cope with this drawback by introducing the concept of sections (of a command file). However, this idea turned out to be very unwieldy and, beginning with version 6.1, we therefore follow a different approach: There is now a clear distinction between commands and parameters. Each command now represents a single procedure.

Another drawback of the older version has been that the user had only limited possibilities to influence the order of command execution. Based on the distinction between commands and parameters there is now a simple solution. Each command is executed as a separate procedure, and commands are executed in the same order as given by the user.

As will be explained in the remainder of this documentation, many more new features have been added to the program. There is no longer a limited focus on transition data, but we try to make the program useful for a broad variety of applications. We also mention that, beginning with version 6.2, the program and its documentation have two authors: Götz Rohwer and Ulrich Pötter.

providing a full statement of the GNU General Public License.

TDA contains some functions, or parts of functions, that have been taken from published sources. Whenever this is the case, this has been explicitly noted in TDA's source code and the documentation.[1] The copyright for these parts is, of course, with their authors.

Finally, we have to add the usual disclaimer. We hope that the program will prove to be efficient and reliable. However, as with all complex programs it is impossible to test its behavior for all possible run-time conditions and environments.[2] Most probably, there are still some errors in the current version of the program. Therefore, we have to add the following disclaimer: TDA is distributed without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose.

Acknowledgments. Many people and institutions have given support to the TDA project. We thank the *Hamburger Institut für Sozialforschung* (Hamburg) for support of the first steps in the development of TDA. The program was further developed as part of the research project *Household Dynamics and Social Inequality*, directed by Hans-Peter Blossfeld, and supported by grants from the *European University Institute* (Florence) and the *European Commission* (Brussels), and then continued at the University of Bremen. We also thank a number of people who have supported the development of TDA, or have given useful comments: Marco Becht, Stefan Bender, Francesco Billari, Hans-Peter Blossfeld, Hannah Brückner, Josef Brüderl, Tak Wing Chan, Juri Ciborra, Sonja Drobnič, Greg Duncan, Christian Dustmann, Beate Ernicke, John Haisken-DeNew, Jan Hoem, Steffen Kühnel, Anders Holm Larsen, Søren Leth-Sørensen, Wolfgang Ludwig-Mayerhofer, John Micklewright, Simon Peters, Rainer Pischner, Ulrich Rendtel, Hilmar Schneider, Gilles Teyssiere, Andreas Timm, Wolfgang Voges, Achim Wackerow. Karl Ulrich Mayer (Max Planck Institut für Bildungsforschung, Berlin) and Hans-Peter

[1] A complex statistical program consists of several different parts. With respect to numerical problems, one can distinguish three stages. First the framework providing the command language to control the program, means to handle input and output of data, and so on. Second, some algorithms that are, in general, also not numerically critical, for instance calculation of log-likelihood functions and their derivatives. And finally some numerically (or for other aspects) critical algorithms, for instance matrix inversion and function minimization. A basic decision has been to take all critical algorithms from published sources in the literature and to document this explicitly in the program's manual.

[2] The current version of the program has been developed and tested on a SUN SparcStation 10/30. Its behavior on other machines has not been tested.

Blossfeld (University of Bremen) have kindly provided some example data (from the *Lebensverlaufstudie*) for many examples in the documentation.

Finally, we should look forward. To become a useful program, the development of TDA needs some response from its users. We would be glad to receive information about problems and bugs and, of course, suggestions for improving the program.

Götz Rohwer
Ulrich Pötter

1.2 Documentation

Development of TDA will be documented in a `history.txt` file and in a User's Manual. The `history.txt` file keeps track of all changes in a chronological order. Whenever we change some part of the program or the User's Manual, and when we add some new features, there should be a corresponding entry in the `history.txt`. This file should be part of each distribution of the program to allow users to update their version of the program and its documentation.

Since TDA is a work in progress, we make its User's Manual something like a loose-leaf book that consists of a set of (more or less) related single documents (URLs in the WWW's HTML jargon). An overview of all available documents can be found in the current table of contents. Identification and ordering of the single documents follow a hierarchical scheme with up to four levels, roughly corresponding to parts, chapters, sections, and subsections. The first page of each document shows its name and the date of its last update. Names of the documents are created as

dl_1.tex	first level
dl_1l_2.tex	second level
d$l_1l_2l_3$.tex	third level
d$l_1l_2l_3l_4$.tex	fourth level

For example, the current document is `d0102.tex`. The extension, `.tex`, shows that we use LaTeX to create the documents. Actually, the distributed files are PostScript files and have the extension `.ps`. We hope that this strategy allows for an easy update of the User's Manual. It should only be necessary to make print outs of new documents, or documents that have been changed since the last release. Information about what is new and what has been changed can be found in the `history.txt` file.

Page numbering is done separately for each document. Headers show the document and page numbers. Since most of the single documents are very short, cross-references are given only by document numbers. Also the subject index uses only document numbers, no page references.

1.3 Installation

TDA is written in the programming language C. Current development is on a SparcStation 10/30 in a UNIX environment. However, since the program is command-driven and only marginally relies on platform-specific features, it is highly portable. The distribution package contains the source code as well as some precompiled versions. Installation is particularly easy if an executable version of the program is already available. With the exception of DOS where the program needs an additional DOS-extender, it will then consist of a single executable file. In order to install the program one simply needs to copy the executable file into a suitable directory of the target machine. Of course, the distribution package might contain the program in a compressed file format and it will then be necessary first to uncompress the file.

DOS. A precompiled version for DOS is created with Watcom's C compiler and called `tda.exe`. In order to be executable it needs an additional DOS-extender. The distribution package contains `dos4gw.exe`, a DOS-extender made available by Watcom as part of its C compiler package. Installation requires to copy both files, `tda.exe` and `dos4gw.exe`, into a directory of the target machine.

The DOS-extender allows TDA to assess extended memory and work as a 32-bit program. This sometimes conflicts with other programs when they try to use the same resources. In any case will it be a good idea to have a look into the file `dos4gw.doc` (a plain ASCII file) that describes several options to re-configure the DOS-extender.

Windows-NT. An executable version of TDA, again compiled with Watcom's C compiler, is available. An additional DOS-extender is not needed. Of course, since TDA is a command-driven program, it must be executed in a suitable shell.

LINUX. The distribution package of TDA contains an executable version for LINUX. The `read.me` file will give information about the version of LINUX and the compiler used to compile the program. If this might not fit with the target LINUX system, it will be be easy to re-compile the program on the target machine (see below).

UNIX Workstations. For most UNIX workstations there will be no

precompiled versions of the program. However, assuming that a C-compiler is available, it will be very easy to create an executable version for the target machine. As part of the TDA distribution package comes an archive that contains the complete source and also a `makefile`. Simply extract all files from this archive, then edit the `makefile` and also have a look into the main include file, `tda.h`, in order to check for some adjustments that might be necessary for the target machine. A final `make tda` will then create an executable version of the program.

1.4 How to Use the Program

TDA is a command-driven program. The normal way to invoke the program is to write, in a suitable shell, the command line

```
tda command1 command2 ...
```

TDA then executes the commands given in the command line and finally terminates. Each command is executed separately, in the order given in the command line (from left to right). If invoked without any commands, the program only shows a short identification message and then terminates.

Some output is always written into the program's *standard output*. Default is the terminal's screen. This standard output can be redirected into an output file by using the syntax

```
tda command1 command2 ... > name_of_an_output_file
```

To append the standard output to an already existing file one can use '>>' instead of '>'. Note that some commands create separate output files to be used for additional output.

Each command given in TDA's command line can be a standard executable command or a special command providing the name of a command file. This special command is

```
cf=name_of_a_command_file
```

A command file is a simple text file containing commands to be executed by the program. A command must begin with a valid keyword identifying the command and must be terminated with a semicolon (;). All text, beginning with the command's keyword until the final semicolon, is interpreted as belonging to the same command. Any text following a # character, until the end of the current line, is interpreted as a comment and not executed.

Both ways of providing commands can be combined. For instance, invoking the program as

```
tda command1 cf=file1 command2 cf=file2 command3 ...
```

would first execute `command1`, then read the command file `file1` and execute the commands found in that file, then execute `command2`, then execute the commands in `file2`, and so on. `cf` commands can also be used inside of a command file. Nesting of `cf`-commands inside of command files is possible up to 10 levels.

Each command begins with a keyword identifying the command that must be given in lower-case letters. Many commands also need or allow for additional parameters. There are basically four syntactical forms.

1. Commands without any parameters:

```
keyword;
```

where `keyword` identifies the command.[1] For example, `time` is a command without parameters.

2. A second syntactical form is

```
keyword = rpar1,rpar2,... ;
```

where `keyword` identifies the command and `rpar1,rpar2,...` are additional parameters that must be separated by commas. As an example, consider the command

```
pdata = fname;
```

In this example, `pdata` is the keyword identifying the command (print data), and the parameter `fname` provides the name of an output file to be used for writing the data.

3. A third syntactical form is

```
keyword (par1,par2,...);
```

where again `keyword` identifies the command and `par1,par2,...` are additional parameters. The brackets indicate that the parameters belong to the same command. They must be separated by commas.

4. Finally, combining the second and third forms leads to the general command structure

```
keyword (par1,par2,...) = rpar1,rpar2 ;
```

[1] When explaining commands we always use a terminating semicolon as required in command files. A semicolon must not be used when the command is given directly in the program's command line.

Box 1 Commands for changing some limitations

`maxnv=...`	maximum number of variables, def. 2000
`maxmat=...`	maximum number of matrices, def. 200
`maxstd=...`	stack size for derivatives, def. 20
`cblen=...`	command buffer length, def. 20000

Most `TDA` commands have this general structure. Note that there are two types of parameters. If parameters have default values they are optional, and when not explicitly given, TDA uses their default values instead. On the other hand, some parameters do not have default values and must be provided by the user. For example, the `fname` parameter for the `pdata` command has no default value; the user must explicitly provide the name of an output file.

If all parameters on the right-hand side are optional, the command can be used as shown in (3); if all parameters inside the left-hand side brackets are optional, the command can be given as shown in (2); and if all parameters are optional, one can simply use the keyword without any parameters.

Upper and Lower-case Letters. TDA is case-sensitive and distinguishes between upper and lower-case letters. Commands and operators must always be given in lower-case letters. There is basically only one exception: variable names must begin with an upper case letter, or with a special character, as will be explained in **2.1**.

Termination. Normally, the program executes one command after the other until execution of the final command is completed. However, each command has a return status indicating whether the command executed successfully. As a default, when a command returns with an error, the program terminates and does not execute any further commands. This default behavior can be changed with the `ierr` command. For instance, given

```
tda ierr command1 command2 ...
```

the program will try to execute all commands regardless of whether some commands return with an error.

Online Use of TDA. There is the possibility to use TDA in such a way that the program remains loaded. This is achieved by invoking the

program as

```
tda i
```

The program then shows up with a prompt (:). Following the prompt, one can input commands. A command can be continued on several lines and must finally be terminated with a semicolon. The command will then be executed and, having given the standard output for that command, the program will show up again with its prompt to await new commands. Termination is with "quit" or "exit".

Limitations. Most commands have some limitations due to the fact that the program stores all data in internal memory and must then know in advance the maximum amount of data to be expected. Some of these limitations can be controlled by optional parameters. For instance, when defining new variables, one can use the `noc` parameter to provide an upper limit for the maximum number of cases (default is 1000).

However, some limitations already occur when the program initializes its basic data structures before executing any commands. In particular, the maximum number of variables, string variables, and matrices cannot be changed after initialization of the basic data structures. Some of these limitations can be modified by using the commands shown in Box 1. These commands must be given in the command line when invoking the program.

1.5 The Example Archive

Command files and data files for all examples used in the User's Manual to illustrate some featurs of TDA are contained in the program's example archive, called `exam.zoo` in the distribution package. One can use the ZOO archive program to list the contents and to extract all or part of the files contained in the archive. With the exception of a few files that illustrate TDA's options for using compressed data archives, all files in the example archive are ASCII text files and can be edited. Most probably, the files will be UNIX text files, meaning that the end-of-line character follows the UNIX convention (that is, a single linefeed character). However, TDA should be able to read data and command files regardless of the end-of-line convention used to create the files.

The example archive contains additional data and command files not explicitly referred to in the User's Manual. In particular, there are several command files that have been used to create the PostScript plots in the manual. They might be interesting for people who want to create similar plots.

The example archive also contains a command file, `test.cf`, that simply consists of a sequence of `cf` commands to call most of the other command files. Running TDA with `test.cf` will then sequentially execute most of the command files contained in the example archive. This might be helpful in order to check whether the program performs correctly.

There is a second example archive, `ehhnew.zoo`, that contains the command files discussed in Blossfeld and Rohwer [1995], updated for the new syntax of TDA.

1.6 Errors and Error Messages

There are two types of errors. First, the user can make errors, meaning that he or she is not using the program correctly. Second, the program can make errors, meaning that the program does not perform as it is expected to do, or does not give proper messages when something goes wrong. In principle, the program should provide useful error messages whenever something goes wrong. However, this is not always the case, and when users find errors or insufficient error messages, they should send us a note.

Since most error messages should be understandable without further explanation, it seems not necessary to list them here. However, we give a few remarks about the main types of errors. (In addition to what is described below, there are some "fatal" error messages indicating serious problems in the program's source code. If such a message occurs, one should always contact the authors.)

Syntax Errors. The most common type of errors is syntax errors meaning that there is some mistake in providing commands to the program. When these errors occur, the user should check the command's definition. As a general rule, the key words for all commands must be given in lower case letters. On the other hand, variable names must begin with an upper case letter in order to be distinguishable from commands. It is also important that each command in a command file is terminated by a semicolon.

Errors in expressions. Expressions are used to define variables, functions, or case select statements. Two types of errors can occur. First, syntax errors, meaning that the definition of an expression cannot be parsed correctly by the program. We have tried to make the syntax for expressions to some degree flexible, but there are some restrictions in using and combining operators. In particular, whenever using variables as part of expressions, they must already be defined.

A second type of error can occur when it is not possible to evaluate an expression, for instance, to calculate the logarithm of a negative or zero argument. These errors will only be detected when actually using the expression. As a general rule, the program will then give an error message (hopefully indicating the type of error) and stop the current

procedure. It is intended that the program's internal status is then the same as at the beginning of the procedure. But most probably this will not always be so.

Errors in reading and writing files. If the program is given a command to read a file (command file or data file), it first tries to open the file. If the file name is not preceded by a path, the file is searched in the current directory. If not found there, the error message will be *can't open ...* Errors can also occur while reading a file; in particular when the file does not contain proper records, or when the record length exceeds the maximum (20000 characters), or when the file contains unexpected characters.

Only two types of errors may occur when writing files. First, when the name given for the output file does not conform to the rules for file names as expected by the operating system where the program is running; and second, when there is not enough space for the file on the disk.

Errors when exceeding internal limits. Although TDA has been developed for use with large models and large data sets there are some limits. Some of these limits can be changed when invoking the program (see 1.4), others can only be changed when compiling the program (almost all definitions can be found in the include file `tda.h`). In any case, the program should give an error message when one of these internal limits is reached.

Insufficient memory. Another type of error occurs when there is insufficient memory. TDA is based on dynamic memory allocation, meaning that, whenever the program requires some memory, this amount of memory is requested from the operating system, and when the memory is no longer needed, it is given back to the system. As a consequence of dynamic memory allocation, memory may be exhausted any time during the execution of the program, not just during its initialization phase. When this happens, the program will give an *insufficient memory* message and stop with the currently performed procedure. Again, it is intended that the program's internal status is then the same as at the beginning of the procedure. But there are chances that this will not always be so.

Errors in numerical algorithms. Finally, errors can occur when executing numerical algorithms. There are three types of problems. First, the program may not be able to execute and finish an algorithm in the

expected way. For instance, this can always happen when maximizing a likelihood with one of TDA's iterative algorithms. The program may not be able to find a (local) maximum, or to determine the maximum with the required accuracy. The program should give, then, an appropriate error message. A second type of problem regards the numerical behavior of an algorithm. This includes rounding errors, numerical overflow and underflow, and numerical instabilities. These problems will be discussed separately. Finally, there are simply unknown errors due to insufficient practical experience with the algorithms as they are currently implemented in the program.

1.7 Some General Commands

There are a few general commands which are sometimes useful. Box 1 provides a summary.

1. TDA is based on dynamic memory allocation, meaning that the program only requests memory from the operating system when this is actually needed; and when this memory is no longer needed, it is given back to the system. Most commands need some memory during execution but it should be no longer allocated when command execution has finished. There are, of course, some exceptions. For example, when defining new variables, memory used for the corresponding data remains allocated until the variables are explicitly removed with the `clear` command (see **2.3**).

   The `mem` command can be used to get information about the currently used memory. Messages about currently used memory do not include the memory used by the operating system to load the program. Only the amount of memory explicitly allocated by the program is shown. This is done first immediately after the program has initialized is basic data structures. The amount of allocated memory shown when the program terminates should be equal. As an additional information when terminating, the program also shows the maximum amount of memory used during execution of commands.

2. The `time` command prints the current date and time into the program's standard output and might be useful to get information about the execution time of commands.

3. If the `ierr` command is given in the command line, the program tries to execute all following commands regardless of whether they return with an error.

4. The `print` command can be used to print some text into the program's standard output. The syntax is

   ```
   print (text) ;
   ```

   Note that TDA always removes blank characters from strings. To keep these characters, `text` should be enclosed in single or double quotation marks.

Box 1 General commands

`mem;`	shows current memory usage
`time;`	shows current date and time
`ierr;`	ignore return status of commands
`print(text);`	print text into standard output
`data;`	info about current data matrix
`data1;`	info about currently available variables
`mach;`	info about machine parameters
`silent;`	controls amount of standard output
`$string;`	invokes a shell to execute string
`help;`	online help

5. The `data` and `data1` commands provide information about the currently defined data matrix. `data` only shows the number of cases and variables; `data1` additionally shows names and definition of variables.

6. The `mach` command can be used to get information about some parameters of the computer where TDA is running. This includes information about largest and smallest floating point numbers and the machine's epsilon (precision). Calculation is based on algorithms adapted from Malcolm [1972] and Gentleman and Marovich [1974].

7. The `silent` command controls the amount of information written into the program's standard and error output. By default, `silent=0`. If `silent=1` nothing is written into the standard output, and if `silent=2` nothing is written into the error output. Finally, if `silent=3`, neither the standard nor the error output is used.

   A further option is provided by `silent=-1`. By default, matrix commands, and also `while`, `repeat`, `if`, and `break` commands, do not echo in the standard output (except when errors occur). While testing command files, it might be useful that also these commands give some echo and this can be requested with `silent=-1`.

8. The command

   ```
   $ string ;
   ```

   invokes a shell in order to execute `string`. For example, `$dir` would show the current directory. This command is most useful when working in the online mode (`tda i`, see 1.4). Note that `string` must be enclosed in quotation marks in order to preserve blank characters.

9. The `help` command is for TDA's online help. This requires that the program has access to the `tda.hlp` file that contains the help information. If given as

   ```
   help;
   ```

   the command shows the entry for the `help` command that explains how to use the command. Otherwise, the command is

   ```
   help [=] string;
   ```

   The command then shows the entry that has a keyword matched by `string`. If there is more than one keyword matched by `string`, the command only shows the available keywords. Note that `string` may contain the wild cards (`.,?,*`).

1.8 Command Flow Control

This section describes a few commands that may be helpful to control the flow of commands in a command file.

Repeat – endrepeat

There are two possibilities to use a repeat–endrepeat construction. The first is:

```
repeat (n = expression);
...
endrepeat;
```

In this construction, `expression` must be a scalar expression that evaluates to a positive integer, say n. The commands in between `repeat` and `endrepeat` are then repeated unconditionally n times.

A second possibility is to use the `repeat` command in the form

```
repeat (n = expression, I );
```

where `I` is an arbitrary matrix name. The command then creates a $(1,1)$-Matrix with this name and while repeating the loop sets `I(1,1)` to the value of the iteration counter.

While – endwhile

For loops that might depend on a condition one can use the following construction:

```
while (expression);
...
endwhile;
```

The commands in between `while` and `endwhile` are repeated as long as `expression` is true, i.e., evaluates to a non-zero value. `expression` can be any standard or matrix expression that results in a scalar value.

If – endif

To check for conditions one can use the construction

```
if (expression);
...
endif;
```

The commands in between `if` and `endif` are executed if `expression` is true, i.e., evaluates to a non-zero value. `expression` can be any standard or matrix expression that results in a scalar value.

Break

Inside `repeat` and `while` loops one can use the command

```
break;
```

All commands following a `break` command are skipped until the next `endrepeat` or `endwhile` command (whatever applies) is reached.

Nesting

The `repeat`, `while`, and `if` commands can be nested up to a level of 100. (This number is defined by `MaxRep` in the header file `tda.h` and can be changed when compiling the program.)

1.9 Programming Repetitions

This section describes a few commands that might be helpful for command repetitions as used, for example, in bootstrap applications or other methods of variance estimation.

Case Selection for Repetitions

There are two related commands. The first one is

```
dblock (mdef=...) = varlist;
```

All parameters are optional. If provided, `varlist` must be a list of comma-separated variable names and the `mdef` parameter must specify a valid matrix name. The command first turns of any possibly active `tsel` (temporary case) selection command. It then creates a data structure that keeps record of blocks in TDA's internal data matrix, defined by the variables in `varlist`. Blocks are defined as contiguous blocks of data matrix rows where the variables in `varlist` have identical values; or, if `varlist` is empty, each data matrix row is treated as a separate block.

The `dblock` command sets the global variable `BNOC` to the number of blocks found in the data matrix. This number is available with the type 1 operators `bnoc` (see 5.2.5.1).

If the optional parameter

```
mdef = M,
```

is used, where `M` is a valid matrix name, the `dblock` command creates a (`NOC`,1)-matrix with the specified name, say `M`, and sets

```
M(i,1) = block number of case (data matrix row) i
```

`NOC` is the global variable that keeps record of the number of currently selected cases (rows) in TDA's internal data matrix.

The second command is

```
repsel (id = V) = S;
```

where S is a matrix expression with dimension (BNOC,1). The command requires that BNOC has previously be defined with the dblock command. Like the dblock command, the repsel command turns off any possibly active tsel (temporary case) selection.

The repsel command works in two different ways depending on whether the optional id parameter is used.

1. If the id parameter is not used, the command creates a new data matrix (actually it creates a filter for the current data matrix) where the cases (rows) in block i ($i = 1, \ldots,$ BNOC) are repeated $\mathtt{S}(i,1)$ times, or omitted if $\mathtt{S}(i,1) \leq 0$.

2. If the id parameter is used, it must specify a valid variable name, say V. The repsel command then creates a new data matrix where block i consists of those cases (rows) where the value of V equals $\mathtt{S}(i,1)$.

Finally note that the repsel command remains active until: (1) a new repsel command is given, or (2) the data matrix is removed, or (3) a tsel command is given, or (4) a dblock command is given, or (5) the command is explicitly turned off with the command

```
repsel = off;
```

Saving Data Matrix Blocks in a Matrix

The comamnd

```
mdefb(B,expression);
```

can be used to save a block of data matrix rows in a matrix. The command requires that the dblock command has been used previously to define data matrix blocks. B must be a valid matrix name, and expression must be a scalar expression that evaluates to a valid block number, say n. The mdefb command then creates a matrix named B and copies block number n from the current data matrix into this matrix.

1.10 User-defined Macros

Defining Macros

A macro can be defined with the command

```
macrodef(MNAME) = { string1; string2; ... };
```

`MNAME` is an arbitrary string to be used as the name of the macro. `string1`, `string2`, ... must be valid TDA commands, or already defined macro names. `MNAME` then becomes a new command, and

```
MNAME;
```

is executed as

```
string1;
string2;
...
```

`string1`, `string2`, ... may contain variable arguments having the predefined names:

```
$1, $2, ..., $100
```

The maximal number of arguments equals the maximal number of macros, defined by the constant `MaxMacro` in the header file `tda.h`. (The value is currently set to 100.) When defining a macro, variable arguments must be contiguous, that is, `$1, $2, $3, ...` Apart from this requirement, they can be used in any way. When invoking the macro name with arguments, TDA uses the following ordering:

```
MNAME(arg_n+1, arg_n+2, ...) = arg_1, arg_2,...,arg_n;
```

`arg_i` is then substituted for `$i` in the macro definition. Arguments can be skipped by using consecutive commas. The corresponding argument in the macro definition will then be empty. (This can be checked with the `exists` operator, see 5.2.5.1.)

Local Matrix Names

Inside a macro definition one can use the command

```
local(A,B,C,...);
```

where `A,B,C,...` are matrix names. The command declares the specified matrix names as being only locally defined. Then, using these names for matrix operations inside the macro will not conflict with identical names that are already defined outside of the macro. The command is ignored if used outside of a macro definition.

Additional Commands

1. In order to check the expansion of macros one can use the command

   ```
   expand = MNAME (arguments) = arguments;
   ```

   This will show the expanded macro based on the provided arguments but will not execute the commands.

2. The command

   ```
   macrolist;
   ```

   gives a list of currently defined macros.

3. The command

   ```
   macroclear;
   ```

   deletes all currently defined macros.

2. Data Handling

Like many other statistical programs, TDA is build around the concept of a rectangular data matrix with rows corresponding to cases and columns corresponding to variables. This data matrix is used for quite different data structures, like cross-sectional data, panel data, event-history data, and relational data. In fact, the concept of a data matrix is quite independent from the interpretation of its contents providing information about a specific type of data. This interpretation is only done by commands when using part of the data matrix as data input.

Part 3 will give more information on different kinds of data structure. The sections in the current part explain how to create and modify TDA's internal data matrix, quite independent of its interpretation as a specific data structure.

2.1 Variables explains TDA's concept of variables as it is used when referring to the internal data matrix.

2.2 Creating New Variables explains the `nvar` command to be used to create and modify a data matrix.

2.3 Removing Variables explains how to remove variables from the current data matrix and how to delete the whole matrix.

2.4 Recoding Variables explains how to recode already existing variables.

2.5 Dummy Variables explains the `ndvar` command that can be used to create dummy variables and interaction effects.

2.6 String Variables discusses a few possibilities to use string variables.

2.7 Namelists explains the concept of namelists that can be used to abbreviate a list of variable names.

2.8 Temporary Case Selection explains the `tsel` command for temporary case selection.

2.9 Writing Data Files explains the `pdata` command that can be used to write the internal data matrix, or selected variables, into an output file.

2.10 Data Archives explains how to create and use compressed data archives.

2.11 Special Format Data Files explains TDA system files and commands that provide an interface to SPSS portable files and to Stata files.

2.12 Data File Utilities describes some commands that can be used to investigate data files.

2.1 Variables

TDA's data matrix is a rectangular array with rows corresponding to cases and columns corresponding to variables. In this context, a variable is defined as a column in the data matrix. Creating a data matrix means to define one or more variables. The data matrix can then be modified by adding and removing variables.

Each variable has a unique name. Variable names are strings of up to 16 characters which may consist of the following letters:

A,. . . ,Z, a,. . . ,z, 0,1,. . . ,9, _, @, $

The first character of a variable name must be an upper case letter, or one of the characters "_", "@", "$". The other characters can be upper or lower case. Variable names are case sensitive, that is, lower and upper case letters are distinguished. The syntax to define variables is:

```
VName <s> [pfmt] (label) = definition,
```

The terminating comma indicates that these expressions are not commands, but parameters which can only be used as part of the `nvar` command that will be explained in 2.2.

`VName` is required and defines the variable's name.

`<s>` is optional and can be used to define a storage size (see below) for the variable's values; the default is `<4>`, that is, values are stored as single precision floating point numbers.

`[pfmt ]` is optional and can be given to define a format statement to be used when writing the variable into an output file. The syntax for `pfmt` is `width.prec`, where `width` is the field width and `prec` the precision. It will be an F format (for example, 0.100) if `width` is positive, or an E format (for example, 1.0E-1) if `width` is negative. The default print format depends on the variable's type and storage size and will be explained in 2.2.

`(label)` is optional and can be used to define a variable label. If defined, the label is (sometimes) used in TDA's standard output. The maximum length is 120 characters. There are no default labels.

`definition` is required and specifies how the variable is to be generated. There are three different possibilities: standard numerical variables, archive variables, and string variables.

1. The syntax for standard numerical variables is

```
VName <s> [pfmt] (label) = expression,
```

where `expression` may consist of numerical constants, random numbers, references to an external data file or to previously defined variables, and operators. How to define expressions will be explained in **5.2**. Here we only mention how to refer to entries in an external data file. This is done by using the special keywords `c1,c2,c3,...` referring, respectively, to the first, second, third, and so on numerical entries in the records of an external data file. For example, `Y=c2`, would define a variable `Y` taking values corresponding to the second numerical entry in the records of an external data file. It is possible directly to combine such references. For example, `S = c1 + c2`, would define a variable `S` with values created as the sum of the first and second numerical entries in the records of an external data file.

When referring to an external data file, it is possible to create a sequence of variables simultaneously. The syntax is

`VName{`n, m`} <s> [pfmt] (label) = c`k`,`

with n and m ($0 \leq n \leq m$) and k ($k \geq 1$) integers. This defines a sequence of $m - n + 1$ variables with names created by appending an integer $i = n, \ldots, m$ to `VName`. The first of these variables refers to the kth entry in the data file, the second refers to the $k + 1$.th entry, and so on. For example, `Y{1,3} = c2`, would be expanded into `Y1=c2`, `Y2=c3`, and `Y3=c4`.

2. Variables can also be created by referring to a compressed data archive. Such variables are called *archive variables*, the syntax is

```
VName <s> [pfmt] (label) = A:AVNAME,
```

where `AVNAME` is the name of a variable in the data archive. Note that the prefix, `A:`, is required to indicate that the following expression is to be interpreted as the name of a variable in the archive. The concept of data archives will be explained in **2.10**.

Box 1 Types of variables

1	if the variable is a string variable.
2	if the variable is a numerical constant.
3	numerical variables which do not (directly or indirectly) involve type 2 operators.
4	numerical variables which (directly or indirectly) involve type 2 operators.
5	if the variable is defined inside of an `edef` command for episode data.

3. TDA's data matrix may only contain data for numerical and for string variables. variables having How to define and use string variables will be explained in **2.6**.

Types of Variables. Each variable has a specific type providing information about how the values of the variable can be generated and, consequently, how the variable can be combined with other variables by operators. Box 1 explains the different types.

There is an important distinction between type 1 and type 2 operators. Type 1 operators can be evaluated by using only information from a single row in the data matrix, or a single record in a data file. On the other hand, type 2 operators need information from other parts of a data matrix. For instance, the `lag` operator is a typical type 2 operator referring to lagged values of a variable. If the definition of a variable contains type 2 operators, its values cannot be generated sequentially but only when all required information is available.

The definition of variable types applies recursively. For example, consider the definition of a variable that does not contain type 2 operators but refers to type 4 variables. It will then, nonetheless, become a type 4 variable.

Storage Size Options. Each data matrix variable is stored according to a specific storage size. Whenever TDA displays the currently defined data matrix variables, the storage size is shown in a column labeled `S`. The possible value are shown in Box 2.

The storage size can be specified individually for each variable in its definition. The default storage size is 4, meaning that values are stored as single precision floating point numbers. The required memory for the data matrix is then $4 \cdot$ `NV` $\cdot$ `NOCMAX`, where `NV` is the number of variables

Box 2 Storage size options

0	values are stored as single bits, range: 0 – 1.
1	values are stored as single bytes, range: +/ − 127.
2	values are stored as two-byte integers, range: +/ − 32000.
4	values are stored as single precision floating point numbers, range: about 7 significant digits.
5	values are stored as four-byte integers, range: 9 significant digits.
8	values are stored as double precision floating point numbers, range: about 15 significant digits.
< 0	negative values are used for string variables. The absolute value of the storage size equals then the length of the string variable.

and `NOCMAX` the maximum number of cases. However, if a variable is defined as a single numerical constant, the storage size is always 8 and only one double precision value, regardless of the number of cases in the data matrix, is allocated in internal memory. One should note that TDA does *not* check whether values of a variable fit into a specified storage size. In particular this may cause problems if missing value codes have to be inserted for variables with a `<0>` storage size.

2.2 Creating New Variables

There is a single command, `nvar`, for creating new variables. If a data matrix does not already exist, the command creates a new data matrix, otherwise the command adds variables to the existing data matrix. Box 1 shows the syntax. Most parameters are optional. Required is at least one `vdef` parameter, that is, an expression of the form

```
VName <s> [pfmt] (label) = definition,
```

to define a variable (see 2.1). Variable names must be unique. A variable name can be used only once on the left-hand side of an expression defining a variable.

Number of cases. If a data matrix does not already exist, one can use the `noc` parameter to set an upper limit for the number of cases. Default is `noc=1000`. If none of the variables refers to an external data file, or to a data archive, TDA will create a data matrix with exactly `noc` cases. If at least one of the variables refers to an external data file (defined with the `dfile` parameter), the number of cases will be $\min\{\texttt{noc}, n\}$, where n is the number of records that can be read from the data file. If one of the variables refers to a data archive file, the default maximum number of cases will be equal to the number of records in the archive data file. The `noc` parameter is only effective if a data matrix does not already exist. The number of cases in an existing data matrix cannot be changed.

External data files. As explained in 2.1, the definition of variables can refer to entries in an external data file by using the keywords `c1,c2,...` It is then necessary to provide the name of the external data file with the `dfile` parameter. The syntax is

```
dfile = name_of_data_file,
```

This parameter can be used several times (maximum is 100). TDA reads all data files sequentially. Of course, the data files should have an identical record structure. Alternatively, variables can refer to a data file in a data archive, see 2.10. Note that variables defined in an `nvar` command must not refer simultaneously to an external data file and to an archive data file. Also, only a single archive data file can be referenced in each `nvar` command.

Box 1 Syntax for `nvar` command

```
nvar (
   vdef,          definition of a variable; this parameter can be used
                  several times to define several variables simultane-
                  ously
   noc=...,       maximum number of cases, def. 1000
   isc=...,       separation character between entries
   dfile=...,     definition of an external data file
   dreclen=...,   fixed length of data file records
   nmrec=...,     number of multiple records, def. 1
   ffmt=...,      format information for data file records
   match=...,     information about matching variables
   isel=...,      case selection while reading data
   vsel=...,      case selection while creating variables
   break=...,     break on condition
   dblock=...,    definition of block mode
   bsel=...,      block mode case selection
   fmt=...,       new print format
   arcdic,        shows value labels for archive variables
   mblnk=...,     new missing value code: blanks, def. -1
   mstar=...,     new missing value code: stars, def. -1
   mpnt=...,      new missing value code: points, def. -1
   mmatch=...,    new missing value code: mismatches, def. -3
   mgen=...,      new missing value code: general, no default
   df=...,        creates an output data file
   keep=...,      keep variables for df option
   drop=...,      drop variables for df option
   bsize=...,     maximum block size, def. 1000
   dtda=...,      TDA description file
   dspss=...,     SPSS description file
);
```

As default, TDA assumes that data file records are terminated by end-of-record characters. This can be a single line-feed character (LF), or a single carriage-return character (CR), or a LF-CR sequence as used by DOS. This allows the records in a data file to have varying length. If all records in a data file have the same length one can use the `dreclen` parameter to specify this type of data file. Given the parameter

```
dreclen = n,
```

with a positive integer `n`, TDA assumes that each record consist of exactly `n` characters, including any end-of-record characters (if present). Whether a data file has fixed record length can be checked with the `lcnt` command, see 2.12.2.

Free and fixed format files. External data files should be plain ASCII files with a free or fixed record structure. Records in a data file may be data records or comments. Empty records consisting of only blank characters, and records beginning with a # character, are interpreted as comments; all other records are assumed to contain valid data. As default, TDA assumes a free format record structure, that is, numerical entries in the data file records are separated by at least one: blank, comma, semicolon, or tabulator character. This then identifies the numerical entries referenced by the `c1,c2,c3,...` keywords.

The `isc` parameter can be used to change the separation character in a free-format data file. The syntax is

```
isc = 'c',
```

where `c` is a specific character.[1] Then, only the character `c` is treated as a separation character and each single occurrence of this character counts as a separate separation character. For example, using the parameter `isc=',`', the record `10,,20` would result in three values: 10, a missing value, and 20. The missing value would be of type `MBlnk`.

Alternatively, there can be a fixed format with, or without, separation characters, meaning that the data in a data file record have fixed locations, beginning at a fixed physical column. It is then necessary to provide information about the location of variables by using the `ffmt` parameter. The syntax is

```
ffmt = c1(n1-m1),c2(n2-m2),c3(n3-m3),...,
```

This means that numerical entries referenced by `c1` begin in physical column `n1` and end in physical column `m1`, and analogously for `c2,c3,...` If a variable occupies only a single physical column, one can use `c1(n1)` instead of `c1(n1-n1)`. Note that it suffices to provide this information for variables which are actually referenced by `c1,c2,...` keywords. It is not necessary to provide additional information about the type of

[1] All characters are used as given, except for the character `t` which is translated to a tabulator character. A blank characters can be specified as `isc=' '`.

numerical entries. Numerical entries can be integers or floating point values. Floating point values can be given in an F format, for example -0.312, or in an E format, for example -3.12e-1. The number of significant digits used depends on the variable's storage size, see 2.1.

Missing values. A data file may contain non-numerical entries to indicate missing values. With free format data files, TDA automatically recognizes two types of missing values: a single point (.) and a single star (*). With fixed format data files, TDA also recognizes blank fields, meaning that the location where to expect a numerical entry consists of only blank characters. In all these cases, the default missing value code is -1, but may be changed with the `mpnt`, `mstar` and `mblnk` parameters, respectively. In all other situations, when TDA cannot correctly read a numerical entry in a data file record, the program stops with an error message. This behavior can be changed with the parameter

```
mgen = value,
```

Given this parameter, TDA substitutes `value` for all numerical entries which cannot be read correctly.

Multiple records. As default, TDA assumes that each physical record in an external data file corresponds to one logical record. There is then a one-to-one correspondence between physical data records and rows of the data matrix. Alternatively, one can define logical records consisting of two or more physical records. Given the parameter

```
nmrec = n,
```

TDA creates logical records by concatenating `n` physical records (default is `nmrec=1`). End-of-record characters are not removed but replaced by blank characters to provide separation characters between numerical entries, see Example 2.

Matching new variables. When a data matrix already exists, the `nvar` command adds the new variables to the existing data matrix. For each row in the existing data matrix, and each new variable, a numerical value will be added to the data matrix.

There are then two possibilities for adding the new variables to the already existing data matrix. As default, TDA assumes *trivial matching*, meaning that values for the new variables are created sequentially for each row in the existing data matrix. If the data matrix contains n rows

and the definition of the new variables refers to an external data file containing less than n records, the variables get the missing value code -3 in the remaining data matrix rows. This missing value code can be changed with the `mmatch` parameter.

Alternatively, one can control the matching of new variables with the `match` parameter; the syntax is

```
match = A1,B1,A2,B2,...,
```

where `A1,A2,...` must be new variables and `B1,B2,...` must be variables in the already existing data matrix. Values for the new variables are then created in such a way that values of `A1` and `B1`, `A2` and `B2`, and so on, are equal. Implied is a one-to-many mapping. Values of the new variables should be unique with respect to `A1,A2,...`; and identical values will be created for all data matrix rows having corresponding `B1,B2,...` values. An example will be given below. Again, the new variables get the missing value code -3 (optionally modified by the `mmatch` parameter) for all data matrix rows without a match with the `B1,B2,...` variables. The variables `A1,A2,...` used in the `match` parameter must be of type 2 or 3 and, in particular, must not contain type 2 operators. Also, the `match` option is not compatible with block mode (see below), and none of the variables must be a string variable.

Case selection. As default, TDA uses all data records from an external data file, or archive data file.

1. The `isel` parameter can be used to select data file records. The syntax is

```
isel = expression,
```

`expression` is then evaluated for each data file record, and a record is only used if the result is not equal to zero. For example, given

```
isel=c1[10,,20],
```

TDA will only use records where the first numerical entry (`c1`) has a value in the range from 10 to 20. Or, given

```
isel=le(rd,0.1),
```

only a 10 % random selection of the input records will be used.

The `isel` expression is evaluated while reading the records from an input data file. Therefore, `expression` can only refer to information that is available in this situation. For external data files specified with the `dfile` parameter, `expression` may only contain numerical constants, random numbers, type 1 operators, already existing variables, and references to data file entries (`c1,c2,c3,...`). When using an archive data file, `expression` may only contain numerical constants, random numbers, type 1 operators, already existing variables, and references to archive variables; these archive variables must be specified in the `nvar` command and must belong to the same data file.

2. The `isel` parameter can only be used to select cases based on information in an external or archive data file; it is not possible to reference variables specified as new variables in the current `nvar` command. To allow for this possibility, one can use the `vsel` parameter with syntax

```
vsel = expression,
```

meaning, again, that only those cases are selected for the data matrix where `expression` results in a nonzero value. In this case, `expression` may contain numerical constants, random numbers, type 1 operators and references to variables of type 2 or 3. The parameter works as follows: For each case in the data matrix, the program first creates values for all new variables, then the `vsel` expression is evaluated based on this data matrix row, and depending on the result, the data matrix row is kept or removed. Note that `vsel` expressions may not contain type 2 operators and, if referring to any of the new variables, these variables should be of type 1, 2 or 3. — Note that the `vsel` parameter can only be used when the definition of at least one of the new variables refers to an archive or external data file. Note also that the `vsel` parameter cannot be used in block mode.

3. The `break` parameter provides a further option. Given

```
break = expression,
```

the creation of new data stops with the first record where `expression` becomes a nonzero value. — Note that the `break` parameter can only be used when the definition of at least one of the new variables refers to an archive or external data file. Note also that the `break` parameter cannot be used in block mode.

Block mode. As default, TDA treats each data file record, and corresponding data matrix row, separately. Alternatively, one can select *block mode* by using the parameter

```
dblock = VName,
```

where `VName` is the name of a variable. TDA then treats each sequence of records (data matrix rows) where `VName` has the same value as one block of records. For instance, if `VName` refers to an ID variable in a data file containing a certain (variable) number of records for each ID value, the records belonging to the same ID number are treated as one block of records.

The effect is that type 2 operators are evaluated differently. As default, type 2 operators are evaluated for all records in the data matrix. For example, `mean(X)` will result in the mean value of variable `X` calculated for all data matrix rows. However, if operating in block mode, type 2 operators are evaluated separately for each block of records; `mean(X)`, for example, will then result in a different mean value for each block of records. More detailed information can be found in **5.2.6**.

The variable used to define block mode must be of type 2 or 3, and that the block mode option is not compatible with the `match` parameter (non-trivial record matching).

If new variables are created in block mode there is an additional option for case selection. The parameter is

```
bsel = expression,
```

At the end of each block of records, TDA evaluates `expression` for each record in the current block and, if the result is zero for at least one record, the whole block is skipped. `expression` may contain numerical constants, random numbers, type 1 operators and references to variables of type 2, 3, and 4. It is thus possible to use type 2 operators for defining variables and then use these variables for the `bsel` expression. For example, assume we want only select individuals with at least five records. One can then first define a variable, say `BN = bnrec`, counting the number of records in each block, and then use the parameter `bsel=ge(BN,5)`. Note that the `bsel` parameter can only be used when the definition of at least one of the new variables refers to an archive or external data file.

While `vsel` and `break` parameters cannot be used in block mode, it is possible to use the `isel` parameter. `isel` expressions are evaluated while reading new records from the input file.

Box 2 Default print formats

storage size	print format
0	2.0
1	4.0
2	6.0
4	0.0
5	11.0
8	0.0

Print formats. Associated with each variable is a specific print format to be used when values of the variable are written into an output file. This print format can be specified in the variable's definition (see **2.1**). If this option is not used the variable gets a default print format depending on its storage size. Default print formats are shown in Box 2, where 0.0 indicates a "free format" meaning that only up to seven significant digits are used.

Since the default storage size is 4, all variables without an explicitly specified print format will get a free format. This can be changed with the `fmt` parameter. All variables having a (default) free format will then get the new print format specified with the `fmt` parameter. The syntax is

`fmt =` $n.m$`,`

where n and m are integers and $m \geq 0$. $|n|$ is the field width, and m is the precision, i.e., the number of digits after the decimal point. If $n \geq 0$ it will be an F format, if $n < 0$ it will be an E format.

Print formats for archive variables are created in the same way as for other variables: If a format information is provided in the definition of a variable, this will be used, otherwise the print format found in the archive's variable description file will be used.

In connection with archive variables one can also use the `arcdic` parameter to request information about value labels. If available in the archive's variable description file, this information is then written into TDA's standard output.

Directly writing output files. Using the `df` parameter, it is possible to write data directly into an output file without creating a data matrix.

Box 3 Example data file `d1.dat`

```
1 1.1 2 3
2 2.2 3 4
3  .  4 *
4 4.4 5 6
```

Box 4 Command file `d1.cf`

```
nvar(
    dfile = d1.dat,      # specifies data file
    X1 = c1,             # define variables
    X2 = c2,
    X3 = c3,
    X4 = c4,
);
pdata = d1;              # print data matrix to output file d1
```

The syntax is

```
df = name_of_an_output_file,
```

Given this parameter, TDA generates data for all new variables but does not store the data in the internal data matrix. Instead, the data are written directly into the output file. This allows to create subsets of very large data files which cannot be kept in internal memory. As default, all variables are written into the output file; optionally, one can select subsets of variables, alternatively with the `keep` or `drop` parameter. Note that this parameter can only be used if a data matrix does not already exist and can only handle variables of type 1 – 4. Furthermore, type 4 variables can only be used if the `nvar` command operates in block mode. If in block mode, this procedure needs memory for at least the maximum number of records per block. The default maximum block size is 1000, but can be changed with the `bsize` parameter.

In addition, one can use the `dtda` and `dspss` parameter to request additional output files describing the generated data. The description conforms to the TDA or SPSS syntax, respectively.

Example 1 Box 3 shows a simple data file. There are four records containing integer and floating point values. Two values are missing. It is a free format data file since all entries are separated by at least one blank character, but also a fixed format data file.

Box 5 Part of standard output from `d1.cf`

```
Creating new variables.

Idx Variable  T S  PFmt  Definition
-----------------------------------
  1 X1        3 4   0.0  c1
  2 X2        3 4   0.0  c2
  3 X3        3 4   0.0  c3
  4 X4        3 4   0.0  c4

Creating a new data matrix.
Maximum number of cases: 1000

Using data file(s): d1.dat
Free format. Separation character(s): default.

Reading data file: d1.dat
Read records: 4

Created a new data matrix.
Number of cases: 4
Number of variables: 4

Missing values  numerical code
------------------------------
Blank        0      -1
Star         1      -1
Point        1      -1
```

Box 6 Command file `d2.cf`

```
nvar(
    dfile = d1.dat,
    X1 = c1,
    X2 = c2,
    X3 = c3,
    X4 = c4,
    ffmt = c1(1),c2(3-5),c3(7),c4(9),
);
pdata = d2;
```

Command file `d1.cf`, shown in Box 4, can be used to create a data matrix containing the data from this data file. The parameters in the `nvar` command first specify the data file and then define four variables corresponding to the four entries in the data file records. The `pdata`

Box 7 Command file `d3.cf`

```
nvar(                    resulting output file d3
    dfile = d1.dat,      ------------------------
    X1 = c1,             1.0  1.1  2.0  3.0  2.0  2.2  3.0  4.0
    X2 = c2,             3.0 -1.0  4.0 -1.0  4.0  4.4  5.0  6.0
    X3 = c3,
    X4 = c4,
    X5 = c5,
    X6 = c6,
    X7 = c7,
    X8 = c8,
    nmrec = 2,
    fmt = 4.1,
);
pdata = d3;
```

command prints the data matrix into an output file.

Box 5 shows TDA's standard output from the `nvar` command in `d1.cf`. A table replicates the definition of variables. The columns labeled `T`, `S`, and `PFmt` show the type, the storage size and the print format, respectively. Some information about missing values is given at the end.

The data file can also be referenced as a fixed format data file. This is illustrated by command file `d2.cf`, shown in Box 6. The two output file, `d1` and `d2`, should be identical.

Example 2 Command file `d3.cf` in Box 7 illustrates the `nmrec` parameter for concatenating physical records. `nmrec=2` means that always two physical records are combined into one logical record. Each logical record then has 8 entries which are referenced by the variables `X1,...,X8`. The right part of the box shows the resulting output file, `d3`. Note that the missing entries in the data file have been substituted by the default missing value code, -1.

Example 3 Command file `d4.cf`, shown in Box 8, illustrates how to add new variables to an existing data matrix. The first `nvar` command creates a data matrix in the same way as was done by command file `d1.cf` in Box 4. The second `nvar` command reads the data file again and defines identical variables, `Y1,...,Y4`. Without using the match parameter this would be identical to defining `Y1 = X1,...` However, the command file uses the parameter `match=Y3,X1`, and the resulting data matrix, shown in the bottom of the box, is therefore created in such a way that these two variables match. Note that there is no match for the

first data matrix row. In this row, all new variables get the missing value code for mismatches, -3.

Box 8 Command file `d4.cf`

```
nvar(                          # creating a new data matrix
    dfile = d1.dat,
    X1 = c1,
    X2 = c2,
    X3 = c3,
    X4 = c4,
    fmt = 4.1,
);
nvar(                          # adding variables
    dfile = d1.dat,
    Y1 = c1,
    Y2 = c2,
    Y3 = c3,
    Y4 = c4,
    match = Y3,X1,             # match Y3 with X1
    fmt = 5.2,
);
pdata = d4;

resulting output file d4
------------------------
1.0  1.1  2.0  3.0 -3.00 -3.00 -3.00 -3.00
2.0  2.2  3.0  4.0  1.00  1.10  2.00  3.00
3.0 -1.0  4.0 -1.0  2.00  2.20  3.00  4.00
4.0  4.4  5.0  6.0  3.00 -1.00  4.00 -1.00
```

2.3 Removing Variables

To remove variables from the current data matrix one can use the command

```
clear = varlist;
```

where `varlist` is a list of variable names separated by commas. All variables specified in `varlist` must be available in the current data matrix. These variables are then removed, that is, all data and data structures associated with these variables are deallocated. Alternatively, one can use the command

```
clear;
```

without any parameters to delete the whole data matrix. One can then begin to create a new data matrix by using `nvar` commands. Note that it is always possible to get information about the currently defined data matrix by using the `data` and `data1` commands, see 1.4.

If `varlist` on the right-hand side of the `clear` command contains namelists (see 2.7), these nameslists and also the variables contained in these namelists are deleted. To remove nameslists, but to keep the contained variables, one can use the command

```
clearnl = list_of_namelists;
```

Then all namelists given on the right-hand side, but no variables, will be deleted.

2.4 Recoding Variables

Variable names must be unique, it is not possible to use the name of an already existing variable inside an `nvar` command to define a new variable with the same name. It is possible, however, to change the values of an existing variable. The command is `recode` with syntax shown in the following box.

```
recode (
    dblock=...,                 definition of block mode
    VName[fmt]=expression,      recode variable VName
    ...                         optionally repeated
);
```

`VName` must be the name of an already existing variable. The values of this variable are changed according to the expression given on the right-hand side. As an option, a new print format can be given in square brackets. Storage size and labels cannot be changed. Also optionally, one can use the `dblock` parameter with syntax

```
dblock = name_of_a_variable,
```

to turn on block mode. If the recode expression contains type 2 operators, they are then evaluated separately for each block. The `recode` command automatically turns off a temporary case selection (see **2.8**) that might be active when the command is executed.

The `recode` command should be distinguished from the `recode` operator that allows to use matrices for recoding variable, see **5.2.6.2**.

2.5 Dummy Variables

Individual dummy variables can easily be defined by using the [.] operator. For instance, `Y = X[3]` would define a dummy variable `Y` taking the value 1 if `X` has value 3, and zero otherwise. For a complete explanation of the [.] operator, see **5.2.5.5**. If one has to create a long series of dummy variables and possibly also interaction effects, it is sometimes more comfortable to use the `ndvar` command with syntax shown in the following box.

```
ndvar(
    Y = dum(X),          create dummy variables
    L = dmul(A,B),       create interaction effects
    ...                  optionally repeated
);
```

Two kinds of parameters can be used (repeatedly). First, the parameter

```
Y = dum(X),
```

can be used to create a set of dummy variables, based on the values of the variable `X`. `X` must be the name of an existing variable, `Y` must be a valid variable name not already used. (The names `X` and `Y` are here only used to illustrate the command; they can be substituted by any other valid variable names.) Given this parameter, TDA creates a separate dummy variable for each nonnegative value of `X`. If `X` has some negative values, an additional dummy variable, `YM`, is created taking value 1 whenever `X` has a negative value. For example, if `X` has values 1, 4, 17, and -1, the command creates four dummy variables: `Y1`, `Y4`, `Y17`, and `YM`. In addition, the command automatically creates a namelist, named `Y`, containing all dummy variables corresponding to nonnegative values of `X`.

Second, one can use the parameter

```
L = dmul(A,B),
```

to create interaction effects. `A` and `B` can be variables or, more general, namelists. `L` must be a valid variable name not already used. This pa-

rameter then creates a set of dummy variables for all possible interaction effects between the variables (in) `A` and (in) `B`. For example, assume that `A = {A1,A2,A3}` and `B = {B1,B2}`. The interaction effects will be

```
L = {A1&B1, A1&B2, A2&B1, A2&B2, A3&B1, A3&B2 }
```

In addition, a namelist containing the dummy variables representing the interaction effects will be created. The names of the dummy variables will be `A1_B1`, and so on. The parameter can be used several times to create higher-order interaction effects. For example, `LL = dmul(L,C)` would create all interaction effects between the variables in the nameslists `L` and `C`. Of course, creating higher-order interaction effects soon exhausts the maximal length of variable names. Automatically created dummy variables always have storage size 0, that is, only a single bit is used to store their values.

The `ndvar` command automatically turns off a temporary case selection that might be active when the command is executed.

2.6 String Variables

TDA's data matrix may contain both numerical and string variables. String variables can become part of the data matrix in one of the following ways:

a) String variables may be imported from SPSS or Stata files.

b) String variables may be present in a TDA data archive.

c) String variables can be defined when reading a standard (ASCII) data file with the `str` operator (see below).

As has been explained in **2.1**, string variables always have a fixed length. This is the absolute value of the storage size which, for string variables, is always a negative integer. There is no specific print format for string variables. The strings are always written with the fixed length.

The `str` Operator. String variables can be defined inside the `nvar` command in the following way:

```
VName = str(n,m),
```

where `n` and `m` are integers. This requires that a data file has been specified with the `dfile` parameter of the `nvar` command. Values of the string variables are then the strings from column `n` up to, and including, column `m` of the data file. There is no limitation to the length of the strings.

One should note that using data files that contain strings as well as numerical entries may result in confusion when the data file is taken as having a free format. The reason is that the counting of numerical entries in the data file records, necessary in order to interpret the `c1,c2,...` references for numerical entries, may become indeterminate. In order to establish determinate references to numerical entries one should use the `ffmt` parameter for fixed format. However, to allow also for a free format interpretation of data files, TDA proceeds as follows. Given the record of a data file, TDA first extracts all strings that are defined with the `str` operator. Then, before references to numerical variables are evaluated, the strings are replaced by blank columns.

Box 1 Data file `d2.dat`

```
1 abc ef 2 ab
2 xyz    3 ac
2 ijklm  4 bc
```

Box 2 Command file `d6.cf`

```
nvar(
    dfile = d2.dat,
    X1 = c1,
    X2 = c2,
    S1 = str(2,8),
    S2 = str(12,12),
    S3 = str(13,13),
);
pdata = d6;
```

resulting output file

```
 1 2 abc ef a b
 2 3 xyz    a c
 2 4 ijklm  b c
```

Using String Variables. Almost all TDA commands require numerical variables. In fact, the only command that currently can use string variables is `pdata`; and this command simply writes the strings into an output file. In order to avoid a check for string variables in all other commands, we employ the following convention: whenever a command expects a numerical variable and is given, instead, a string variable, the value of the string variable is taken to be zero.

Operators for String Variables. There are, however, a few operators that can be used to create numerical variables from string variables. The available operators are described in **5.2.5.9** and **5.2.6.8**.

Example 1 To illustrate string variables, command file `d6.cf` (Box 2) uses the data file `d2.dat` shown in Box 1. The data file contains numerical entries and strings. The `nvar` command defines first two numerical variables, `X1` and `X2`, by referring to data file entries `c1` and `c2`. (Remember that strings are replaced by blank characters before such references are evaluated.) Then follow three string variables. The resulting output file created by the `pdata` command is shown in the same box.

2.7 Namelists

The word *namelist* is used to denote a comma-separated list of variable names. Namelists can be defined with the `nlist` command. The syntax is

```
nlist (LName = V1,V2,...);
```

where `LName` is the name used to refer to the list of variables given on the right-hand side. Then, in most TDA commands, one can use the short name of the namelist instead of writing the complete list of variable names. The syntax for the names of namelists is the same as for variables. Up to 50 namelists can be defined. However, only a single namelist can be defined with each `nlist` command. The `data1` command shows the currently defined nameslists (if any).

Namelists can be deleted in the same way as variables with the `clearnl` command, see 2.3. Note also that namelists can be combined with variable names. For example, if `Y` and `Z` are names of variables and `NList` is the name of a namelist, it would be possible to request a linear regression with the command

```
lsreg = Y,NList,Z;
```

The dependent variable would be `Y` and the set of independent variables would consist of the variables in `NList` plus `Z`.

It is possible to define a new namelist using already defined namelists. For example,

```
nlist (NList1 = Y,NList,Z);
```

would define another namelist consisting of `Y`, followed by the variable names in `NList`, and then `Z`. The ordering of variable names is kept.

2.8 Temporary Case Selection

TDA's internal data matrix always has a fixed number of cases (rows) that is determined when the data matrix is created for the first time. Therefore, the number of cases actually contained in the data matrix cannot be changed afterwards. As default, all TDA procedures which require data use all cases in the currently defined data matrix. However, one can request a temporary case selection by using the command

```
tsel = expression;
```

TDA then creates an indicator variable by evaluating `expression` for all cases in the current data matrix, and all subsequent procedures will only use those cases from the data matrix where `expression` has a nonzero value. A request for temporary case selection remains active until one of the following conditions occur:

- A new `tsel` command substitutes the previous one.
- The current case selection is explicitly turned off with the command

```
tsel = off;
```

- It follows a command that automatically turns off any active temporary case selection (`nvar`, `ndvar`, `recode`).

The `tsel` command implies deallocation of any data structures which depend on case selection. In particular, data structures for episode, sequence and relational data will be deallocated if there is a request for a new temporary case selection. The `tsel` command also implies a re-evaluation of case weights, if defined, see **6.1.2**.

Case Selection with a Matrix Expression

The usage of `tsel` described above requires a standard (scalar) expression on the right-hand side. It is possible, however, to use the `tsel` command in the following way:

```
tsel = matrix_expression;
```

It is required that the matrix expression (or simply a matrix), say S, has dimensions NOCDM $\times$ 1 where NOCDM is the number of cases in the data matrix. (This is available from the nocdm operator, see 5.2.5.1.) The command then selects all those rows i from the current data matrix where the value of $S(i)$ is not equal to zero.

2.9 Writing Data Files

Data contained in the internal data matrix can be written into an output file. The command is `pdata`, the syntax is shown in Box 1. All parameters are optional. Without any parameters, the command is simply

```
pdata = fname;
```

where `fname` is the name of an output file. TDA will then write the data for all variables contained in the current data matrix into this output file. If a temporary case selection is active (see 2.8), only the selected cases will be used. For writing the data, the program uses the print format associated with each variable. By default, following each numerical entry, a blank character is inserted allowing to use the data file as a free format input file.

1. To select a subset of variables one can use the parameter

```
keep = varlist,
```

where `varlist` is a list of variable names separated by commas. Only the variables specified in `varlist` will then be written into the output file. Alternatively, one can use the parameter

```
drop = varlist,
```

Then, only variables which are not specified in `varlist` will be written into the output file.

2. By default, the ordering of the records in the output file will be equal to the ordering of the rows in the data matrix. Alternatively, one can use the `sort` parameter to sort the data matrix before variables are written into the output file. The syntax is

```
sort = S1,S2,...,
```

where `S1,S2,...` are names of variables. The data matrix is then sorted in ascending order first with respect to `S1`, then with respect to `S2`, and so on. Note that string variables cannot be used for sorting. Of course, one can first create a numerical variable that sorts the strings with the

Box 1 Syntax for `pdata` command

```
pdata (
     keep=...,        keep specified variables
     drop=...,        drop specified variables
     sort=...,        sort according to specified variables
     noc=...,         write only first noc cases
     nn = n1,n2,      use only rows n1 – n2
     transp,          transpose data matrix
     nc = n,          create a new record after each n variables
     nq = n,          concatenate n records, def. 1
     ap = 1,          append data to output file, def. ap = 0
     l0 = ...,        controls printing of numerical values, def. 0
                      0 : leading blanks remain empty
                      1 : fills leading blanks with zeros
     sepc = ...,      separation character, def. blank
                      sepc=none : no separation character
                      sepc=c : use c as separation character
     prn =...,        option for writing single variables, def. prn = 0
                      1 : write a variable as triangle matrix
                      2 : write a variable as square matrix
     dtda=...,        create a TDA description file
     dspss=...,       create an SPSS description file
) = fname;
```

help of the `strsp` operator, and then use this variable to sort the output file.

3. By default, all cases contained in the data matrix are written into the output file. When using the parameter

`noc = n,`

only the first `n` data matrix rows are written into the output file. Alternatively, one can use the parameter

`nn =` n_1, n_2`,`

Then, only data matrix rows $i = n_1, \ldots, n_2$ are written.

4. As an option one can use the parameter `transp` to request that the data matrix is transposed before variables are written into the output file.

Box 2 Command file `d5.cf`

```
nvar(                              output file
    dfile = d1.dat,                -----------
    X1 = c1,                        1.0  2.0
    X2 = c2,                        1.1  3.0
    X3 = c3,                        2.0  3.0
    X4 = c4,                        2.2  4.0
    fmt = 4.1,                      3.0  4.0
);                                 -1.0 -1.0
pdata (                             4.0  5.0
    keep = X1,X3,X2,X4,             4.4  6.0
    nc = 2,
) = d5;
```

5. When creating a data file with the `pdata` command it is possible to replace each data matrix row by several records in the output data file. The parameter is

```
nc = n,
```

where n is an integer, $n \geq 1$. Then a newline character is inserted after each n values written into the output file, see Example 1.

6. Alternatively, one can use the `nq` parameter. If `nq = n`, then each consecutive set of `n` rows of the data matrix is written as a single record into the output file.

7. By default, values of variables are separated by one blank character in the output file. This may be changed with the `sepc` parameter. Also, by default, numerical entries are right justified and leading columns remain empty. The `l0=1` parameter can be used to request that leading columns are filled with zeros.

8. By default, the `pdata` commands creates a new output file. An already existing file with the same name is overwritten without warning. When using the `ap=1` parameter (default is `ap=0`), the data is appended to an already existing file.

9. The `dtda` parameter can be used to request an additional output file containing a TDA description of the data file. The syntax is

```
dtda = name_of_an_output_file,
```

The output file contains an `nvar` command that can be used to read the data file in a subsequent call of TDA. In a similar way, one can use the

parameter

```
dspss = name_of_an_output_file,
```

to request an additional output file that can be used to read the data file with SPSS or, after some editing, with SAS. Note that the `dtda` and `dspss` parameters are ignored when using the `transp` option.

10. The `prn` parameter can be used when writing a single variable into an output file. If `prn=1`, the variable is written as a triangle matrix, if `prn=2`, it is written as a square matrix. This option is mainly used to change the format of relational data which, in TDA, are assumed to be given as a single column vector (variable) but in other programs are often required in the format of an adjacency matrix (or part of it).

Example 1 Command file `d5.cf`, shown in Box 2, illustrates the `nc` parameter. The command file uses data file `d1.dat` (see **2.2**). We want an output file where each data matrix row is replaced by two data file records. The first record shall contain variables `X1` and `X3`, the second record `X2` and `X4`. We therefore use the keep command to change the order of variables, and then use `nc=2` to create two records for each data matrix row.

2.10 Data Archives

In empirical research, in particular when working with longitudinal data, one often has to handle very large data sets. Given this situation, it would be nice if the data could be stored and directly accessed in a compressed way. This chapter describes how this can be done with TDA, based on Rahul Dhesi's program ZOO. This is a general purpose archive program to generate and manage an arbitrary set of files into a compressed archive. Files can be added, deleted, and updated; also some options to repair damaged archives are available.

ZOO is similar to other archive programs, for instance PKZIP, ARC, and LHARC. To be used with TDA, we have chosen ZOO because it is highly portable, already running on many different computer platforms, and is freely available. (Rahul Dhesi, who owns the copyright of ZOO, has made his program publicly available without any license fee. So it is possible to distribute his program together with TDA as an additional data management utility.) Moreover, ZOO incorporates very efficient compression algorithms achieving 80 – 90 percent compression rates with typical data sets used in empirical research.

Of course, to be useful one not only needs a way to create compressed archives but, in addition, have direct and selective access to the data stored in the archive without the need of prior extraction of all involved data files. This capability has been build into TDA. So one can use ZOO to build and manage data archives and then use TDA to access the compressed data. Details are given in the following sections.

2.10.1 Creating Data Archives explains how to create a compressed data archive to be used with TDA.

2.10.2 Variable Description Files explains how to create a description of the variables contained in a data archive.

2.10.3 Archive Description Files explains how to create a description of data archives.

2.10.4 Data Archive Commands summarizes the steps required to create a data archive and explains how to use such archives with TDA.

2.10.1 Creating Data Archives

Creating a data archive is done with ZOO. Like TDA, ZOO is a command-driven program. Quite a large number of commands can be used to control the program; a full description can be found in the program's manual that is always distributed along with ZOO. Here we only explain some basic commands.

Generally, the syntax for using ZOO is

```
zoo commands [archive_name] [file1 file2 ...]
```

`archive_name` is the name of the file to be processed by ZOO as an archive. Usually, an archive name has the extension `.zoo`; this is the default, assumed by ZOO if a file name without an extension is given. `commands` is a string of letters interpreted by ZOO as a command, or a set of commands. The letters are case sensitive. The following examples will explain some basic options.

`zoo h`	displays a short list of commands to be used with ZOO on the system's standard output. Using an upper case `H`, instead of `h`, gives an extended description.
`zoo l exam.zoo`	lists the contents of the archive `exam.zoo`. As already noted, the name of the archive can be given without an extension: `zoo l exam`.
`zoo ah exam fname`	adds the file `fname` to the archive `exam` using a high compression mode.
`zoo x exam fname`	extracts the file `fname` from the archive `exam.zoo`. If no file name is given (`zoo x exam`) all files contained in the archive are extracted.
`zoo D exam fname`	deletes the file `fname` from the archive `exam.zoo`. In fact, the file is not deleted but kept as an earlier version in the archive. To definitively delete the file, the archive must be re-packed.
`zoo P exam`	rebuilds the archive `exam.zoo` with all background files actually deleted. The original archive is saved as `exam.bak`.

Box 1 Example data files.

```
  adat.1        adat.2        adat.3     adat.p
----------    -----------    ---------   -------
1 -1.01 90    1 1 1.01 -1    90 9001 1     1 90
6 -1.06 95    2 6 1.06 -2    91 9003 3     2 90
2 -1.02 90    3 2 1.02 -3    95 9008 8     3 91
7 -1.07 95    4 7      -4    97 9009 9     4 91
3 -1.03 91    5 3 1.03 -5                  5 91
8 -1.08 95    6 8 1.08  *                  6 95
4 -1.04 91    744 1.04-77                  7 95
9 -1.09 97    8 9      -8                  8 95
5 -1.10 91                                 9 97
```

Box 2 Table of contents for archive `exam.zoo`

```
Archive examz.zoo:
Length    CF  Size Now  Date      Time
--------  --- --------  --------- --------
     108  48%       56   3 Nov 55 20:35:44+457279   adat.1
      96  33%       64   3 Nov 55 20:35:44+457279   adat.2
      40  20%       32   3 Nov 55 20:35:44+457279   adat.3
      45  27%       33   3 Nov 55 20:35:44+457279   adat.p
--------  --- --------  --------- --------
     289  36%      185     4 files
```

As an example, used also below to explain TDA's commands to work with ZOO archives, we will take the four data files shown in Box 1. The ZOO archive will be called `examz.zoo`; it is generated with the command

```
zoo ah examz.zoo adat.*
```

Looking at the result with

```
zoo l examz
```

shows a table of contents (Box 2). Any kind of files can be put into a ZOO archive. However, our main interest is in data files containing variables which can be accessed selectively. Consequently, to be useful with TDA, all data files in an archive should have a record structure.

A record structure can be given in two different ways. First, records can be defined by a fixed record length. It is assumed, then, that all records in a file have the same length to be defined in the archive description file (see below). Second, the records in a file may have variable

length. In this case, the end of each record must be given explicitly by an end-of-record character.

End-of-record (newline) characters are not standardized across different platforms. On UNIX machines it is a single line-feed (`0a`, hexadecimal); on Macintosh computers it is a single carriage return (`0d`, hexadecimal); and DOS uses two characters: carriage return and line-feed. All three possibilities may be used for files with variable length records contained in an archive. However, when the program *writes* an end-of-record (newline) character it is always the specific one for the machine where the program is running.

One should note that TDA only supports archives with data files containing variables in a fixed-field format. This restriction is reflected by the requirement that for all variables their offset as well as their field length must be explicitly specified (see **2.10.2**). This requirement also applies to variable length records. In this case, if a variable is defined with an offset out of the actual length of a record, the record is filled up with blank characters resulting in missing values for these variables.

2.10.2 Variable Description Files

In order to access variables contained in the archive's data files, these variables have to be described. Otherwise it would be impossible to recognize that a file actually contains data organized as variables. TDA is told about variables by means of a *variable description file*.

A variable description file is a standard ASCII text file. Each line of this file describes a variable, or is a comment line if its first non-blank character is a # character. A line describing a variable consists of five fields, the last one is optional. Fields are taken to be in free format, separated by at least one blank character. However, the first field must begin in the first column of a record.

1. The first field is the *name of the variable*. The number of characters of a variable's name is not fixed. It is proposed, however, that a maximum of 16 characters is used. The characters can be upper and lower case letters, digits, and the underline character. The first character must be a letter. Note that variable names are case sensitive. If retrieving variables from a compressed data archives, the variable names must be exactly the same as defined in the variable description file.

2. The second field must be a numerical entry and is interpreted as the *ID number of the data file* where the variable is to be found. It must be the same number as assigned to the data file in the archive description file (see **2.10.3**).

3. The third field is again numerical and is interpreted as an *offset in the data file record* were the value of the variable begins. Counting begins with zero as the first column in a data file record.

4. The fourth field is also numerical and, for numerical variables, may be given as `n` or `n.m` with `n` and `m` integers. `n` must be greater than zero and is interpreted as the field width for the values of the variable. The syntax `n.m` should be used to provide a print format for floating point values; see the description of the `arcv` command in **2.10.4**.

Data archives may contain string variables. These are identified by a single negative integer in the fourth field of the variable description file. The absolute value of this entry is then taken as the length of the string

Box 1 Variable description file for archive `examz.zoo`

```
# examz.var variable description file for examz.zoo

A1    1        0        2     first variable of dat.1
  This is some additional information
  about the first variable.

A2    1        3        5.2   second variable of dat.1
A3    1        9        2     third variable of dat.1

B1    2        0        1     first variable of dat.2
B2    2        1        2     second variable of dat.2
B3    2        4        4.2   third variable of dat.2
B4    2        8        3     fourth variable of dat.2

C1    3        0        2     first variable of dat.3
C2    3        3        4     second variable of dat.3
C3    3        7        2     third variable of dat.3

  Additional information may be given as value labels
  (1) = label ...
  (3) = label ...
  (8) = label ...
  (9) = missing value

P1    4        0        1     first variable of dat.p
P2    4        1        3     second variable of dat.p
```

variable.

5. The last field is optional and can be an arbitrary string of characters that describes the variable as a variable label. Different from the other items, this field may contain blank characters. If it is present it starts with the first non-blank character after the fourth field up to the end of the current line.

TDA has no provision to use value labels in its printouts. However, such information may be provided in a variable description file simply for informational purposes. To do so, the following convention is adopted. Whenever a non-comment line in a variable description file begins in the first column of a record it is taken as the description of a variable as described above. All non-comment lines beginning with at least one blank character are taken as additional information about the preceding variable. This additional information is not interpreted by TDA but is

shown when variable descriptions are printed and the `arcdic` command is given.

A variable description file, `examz.var`, for the example data archive `examz.zoo` is shown in Box 1. One should note that a single variable description file is used to describe *all* variables in an archive. It is required, therefore, that variable names are unique across all data files. On the other hand, it is not required that all variables, actually contained in an archive, are explicitly described. Of course, only variables described in the variable description file can be used by TDA. The ordering of the variable descriptions is not important.

Having created a variable description file, it must be added to the archive. For our example archive, we can do this with the command

```
zoo ah examz examz.var
```

2.10.3 Archive Description Files

A data archive contains one or more data files and a single variable description file. In addition, TDA needs some information about the contents of the archive. This information is contained in an *archive description file*. This is again a standard ASCII text file, similar to a TDA command file. Empty lines, or lines where the first non-blank character is a # character, are treated as comments and ignored. All other lines are interpreted by TDA.

The first line that is not a comment is always interpreted as the name of an archive, possibly headed by a path where to find it. All other non-comment lines are interpreted as descriptions of files contained in the archive. It is not necessary to describe all files contained in the archive; however, only files described in the archive description file are recognized by TDA. The maximum number of files that can be described for a single archive is set to 200 in the standard compilation procedure for TDA.

For each file, six pieces of information must be present, given in six fields separated by one or more blank characters:

1. The first field is the *ID number of the data file* described in this line of the archive description file. It must be a non-negative integer number, unique for each of the files. The ordering is not important. This number is taken as the ID number of the data file, and it must be the same number as used for attaching variable names to data files.

2. The second field is the *name of the file*. It must be the same name as used in the creation of the ZOO archive and should conform to the operating system's requirements for file names.

3. The third field is a numerical entry and is interpreted as the *type of the file*. Currently, only two types are distinguished: If the value is 1 the file is assumed to be a data file. If the value is 2 the file is taken to be the variable description file of the archive (see **2.10.2**). Each archive description file must contain exactly one description of a variable description file.

4. The fourth field is a numerical entry and is interpreted as the *record length of the file*. It must be a non-negative integer. If its value is greater than zero it is assumed that the file consists of records of equal length

Box 1 Archive description file for archive `examz.zoo`

```
# examz.des
# archive description file for examz.zoo

  examz.zoo

# N  Name      Type  RecLen  Records    NVar

  1  adat.1      1       0        9       3
  2  adat.2      1      12        8       4
  3  adat.3      1      10        4       3
  4  adat.p      1       5        9       2
  9  examz.var   2       0       29       0
```

given by this value. (If the records have one or more end-of-line characters they should be included in the record length.) If the value is zero, it is assumed that the file has variable length records. In addition, it is then assumed that each record is terminated by an end-of-line character. Normally, data files will have fixed length records, while variable description files will have variable length records. However, also data files may have variable length records, if only the variables that are described in the variable description file can be found at fixed positions in the records.

5. The fifth field is a numerical entry and is interpreted as the *number of records of the file*.

6. The sixth field is a numerical entry and is interpreted as the *number of variables* contained in the file. Currently, this value is used for informational purposes only. In the case of a variable description file this entry should be set to zero.

Box 1 shows as an archive description file, `examz.des`, for the archive `examz.zoo`. The first non-comment line contains the name and optionally the path of the ZOO archive. The following lines describe the files contained in the archive. The ordering is not important. In this example, there are four data files of type 1, and one variable description file of type 2. The data files are defined with a fixed record length, the variable description file has variable length records.

Notice that the definition of record length always includes any end-of-record characters. This may result in differences between DOS and UNIX systems. The above example was created on a UNIX system where only one character is used for the end of a record in a text file. However, ZOO

data archives are portable across different computer platforms without any modifications;[1] and also the same archive description files can be used.

[1] Of course, ZOO archives must be transferred as *binary files* between different computers.

2.10.4 Data Archive Commands

We begin with a short summary of the steps required to create a compressed data archive. It is assume that there is a set of ASCII data files with a fixed format record structure.

1. Create a ZOO archive containing the data files. This is done directly with the ZOO program. Note that one should always use ZOO's `ah` option to add files in the high compression mode.

2. Choose ID numbers for all data files in the archive and create a variable description file describing all variables in the archive. (If the data files are available as SPSS portable files, this can be done automatically with the `rspss` command, see 2.11.2.) Check the file with the `arcvc` command and be sure that it does not contain errors. Then add the variable description file to the archive using again the `ah` option for high compression mode.

3. Create an archive description file, check the archive with

```
tda arcd=... arcc
```

and check the output. If TDA does not find any errors, the archive can be used for data retrieval.

Having created a data archive, it can be used with TDA to retrieve data. The basic command is `nvar` as described in 2.2. One simply uses the `nvar` command and specifies the required archive variable using the syntax:

```
VName = A:AVNAME,
```

where `AVNAME` is a variable name contained in the archive's variable description file and `VName` is a corresponding variable name to be used by `TDA`. It would also be possible to use the extended syntax

```
VName <s>[pfmt](label) = A:AVNAME,
```

but TDA automatically tries to find an optimal storage size and print format for archive variables, based on information in the variable description file. However, TDA is only able to retrieve archive variables if

given the name of an archive description file. The command is[1]

```
arcd = name_of_an_archive_description_file;
```

Given this command, TDA reads the archive description file and tries to open the archive for subsequent reading. Having successfully opened an archive, the connection remains valid until it is explicitly turned off with the command

```
arcd = off;
```

or until another archive is opened with a new `arcd` command. This allows to use several different archives in a single command file. It also allows to assess the same archive with a series of `nvar` commands. This is, in fact, necessary because each separate `nvar` command can only retrieve variables from a single archive data file. It is required to explicitly match variables contained in different archive data files.

In addition to the basic `arcd` command to open a data archive, there are three commands to support working with archives.

1. The `arcc` command, without any parameters, can be used to check whether TDA can successfully read the files in an archive. Given this command, TDA tries to read all files in an archive and compares the results with the information in the corresponding archive description file. A message about any discrepancies is given in the standard output. In addition, the command checks whether all variable names in the variable description file are unique. To illustrate, we check our example archive with

```
tda arcd=examz.des arcc
```

The result is shown in Box 1.

2. To check a variable description file one can use the command

```
arcvc (df=fname) = name_of_an_variable_description_file;
```

TDA then reads the variable description file and checks for correct syntax and unique variable names. The result is shown in the standard output. For example, the command

```
tda arcvc=examz.var
```

[1] If the `arcd` command is used without any arguments, it provides information about the currently defined data archive (if any).

Box 1 Output from `arcc` command.

```
> arcd=examz.des
-------------------------------------------------------
Reading archive description file: examz.des
ZOO data archive: examz.zoo
Checking definition of files in archive.

FN  Type  RLen  Records  NVar    Size    M  Name
-------------------------------------------------
  1   1     0        9     3       108  2  adat.1
  2   1    12        8     4        96  2  adat.2
  3   1    10        4     3        40  2  adat.3
  4   1     5        9     2        45  2  adat.p
  9   2     0       29     0       911  2  examz.var

> arcc
-------------------------------------------------------
Archive check. Reading files defined in: examz.des

File:   1  adat.1    records:       9
File:   2  adat.2    records:       8
File:   3  adat.3    records:       4
File:   4  adat.p    records:       9
File:   9  examz.var records:      29
No errors found.
```

should show that the variable description file for our example archive is correct and that all variable names are unique. The optional parameter `df=fname`, with `fname` the name of an output file, can be used to create a variable description file with unique variable names. If the input file uses variable names more than once, these names are made unique by appending the corresponding logical file number. Note that this procedure assumes that variable names corresponding to the same logical file number in the input file are already unique. This is not checked, but might be checked by using the `arcvc` command a second time with the output file created by the `df` parameter.

3. The command

```
arcv (fn=f,arcdic) = fname;
```

can be used to get information about the variables contained in a data archive. All parameters are optional. If used without parameters, the command prints a list of all variables contained in the data archive into

Box 2 Output from `arcv(fn=adat.2)` command

```
# arcd = examz.des;
# nvar(
# B1   <1>[2.0] = A:B1   , # [adat.2] first variable of dat.2
# B2   <1>[2.0] = A:B2   , # [adat.2] second variable of dat.2
# B3   <4>[5.2] = A:B3   , # [adat.2] third variable of dat.2
# B4   <2>[3.0] = A:B4   , # [adat.2] fourth variable of dat.2
# );
```

the standard output. The printout follows the syntax of the `nvar` command and can be used as a command file for retrieving variables from the archive. If the name of an output file, `fname`, is given on the right-hand side, this file is used instead of the standard output. The `fn` parameter can be used to specify data archive files. It can be used more than once. If used, only variables contained in the specified data file(s) are written. The `arcdic` parameter can be used to request that also value label information (if present in the archive's variable description file) is written into the output file. As an illustration, the result of

```
tda arcd=examz.des arcv(fn=adat.2)
```

is shown in Box 2.

Creating a storage size and a print format is based on the format information in the variable description file. If the format information is given by a single integer, n, it is assumed that the variable takes only integer values. Also if the format is given as $n.m$ and $m = 0$. Otherwise, it is assumed that the variable has floating point values.

In both cases, calculation of a storage size is based on $n^* = \max\{n, 2\}$. In the case of integers, the storage size is 2 if $2 \leq n^* \leq 4$ and otherwise 5. For floating point numbers the storage size is 4 if $n^* \leq 7$ and otherwise it is 8. The print format is $w.m$ with $w = n^*$ for integers and $w = \max\{n^*, m + 3\}$ for floating point numbers. Note that the `nvar` command needs the `afmt` parameter to enable these conventions. If the `nvar` command is used without the `afmt` parameter, archive variables are treated like all other variables.

2.11 Special Format Data Files

This chapter contains the following sections.

2.11.1 TDA System Files explains how to use system files for TDA's internal data matrix.

2.11.2 SPSS Files explains how to read and write SPSS portable and sav files.

2.11.3 Stata Files explains how to read and write Stata files.

2.11.1 TDA System Files

TDA system files are used to save the current data matrix in a file, or to restore the data matrix from such a file. Since a system file stores data in a machine-dependent binary format, reading the data from a system file is much faster than creating an internal data matrix from external data or archive files. The command to save the data matrix in a system file is

```
wsys = name_of_an_output_file;
```

The right-hand side is optional. The default system file name is `tda.sys`. Given this command, TDA write the contents of the currently defined data matrix into the output file; any currently active temporary case selection is ignored. String variables and type 5 variables are not written. Also information about case weights, if defined, is not written into the system file.

To restore a data matrix from a system file, the command is

```
rsys = name_of_a_system_file;
```

Again, the file name is optional and the default name is `tda.sys`. Given this command, TDA tries to restore a data matrix from the specified input file. A list of variables read from the system file is written into the standard output. Note that this command can only be used if a data matrix does not already exist. Otherwise, one should first execute a `clear` command (see 2.3).

A TDA system file always begins with some plain ASCII records, then follow the data in a machine-dependent binary format. The structure is as follows:

1. The first record contains an identification string: `TDA System File (n)`, with `n` the version number, followed by the date and time when the file was created.

2. Then follows the number of cases and variables, and a description of all variables in plain ASCII.

3. Finally, for each variable there is a single binary record containing the data.

2.11.2 SPSS Files

This section describes commands that can be used to read and write SPSS portable and sav files.

2.11.2.1 SPSS Portable Files describes commands for reading and writing SPSS portable files.

2.11.2.2 SPSS Sav Files describes commands for reading and writing SPSS (for Windows) sav files.

2.11.2.1 SPSS Portable Files

There are two commands, `rspss` and `wspss`, to read and write SPSS portable files. The SPSS *Base System Syntax Reference Guide* (1992, p. 234) gives the following explanation: "A portable data file is a data file created by SPSS and used to transport data between different types of computers and operating systems [...] or between SPSS, SPSS/PC+, or other software using the same portable file format. Like an SPSS data file, a portable file contains all of the data and dictionary information stored in the working data file from which it was created." Implementation in TDA is based on: *SPSS for Unix. The PPF Portable File Format, Release 4.0*, published by SPSS Inc. in April 1990.

Reading SPSS portable files. The command is `rspss` with syntax shown in Box 1. It is expected that the name of an SPSS portable file is given on the right-hand side; all other parameters are optional. TDA then tries to create a data matrix based on the contents of the SPSS file or, if the `df` parameter is used to give the name of an output file, the data are directly written into this output file without creating an internal data matrix. Note that, in both cases, the command is only executed if a data matrix does not already exist. Note also that one can use `dfa`, instead of `df`, in order to append the data to an already existing files.

1. The input file is expected to be a plain ASCII file consisting of records each having 80 characters (bytes) *plus* end-of-record characters (a single line-feed (UNIX) or a carriage return – line-feed sequence (DOS). If a record contains less than 80 characters it is filled with blank characters.

If the input file does not contain end-of-record characters, one can use the `len` parameter with syntax

```
len = n,
```

where `n` is one of the integers 80, 81, or 82. TDA then assumes that each record consists of exactly `n` characters (bytes).

2. Basic information about variables in an SPSS portable file is given by the variable's name and, optionally, by variable and value labels. TDA uses the same variable names as found in the SPSS file. If present, also the variable labels are used. Values labels are ignored.

Box 1 Syntax of `rspss` and `wspss` commands

```
rspss (
    len=...,          record length, def. 80
    noc=...,          maximum number of cases
    msys=...,         system missing value code, def. -5
    fmt=...,          new print format
    df=...,           write data directly to output file
    dvar()=...,       create/update variable description file
    arcd()=...,       create/update archive description file
) = file_name;

wspss (
    keep=...,         keep variables
    drop=...,         drop variables
    sort=...,         sort cases
) = file_name;
```

3. Since SPSS portable files do not contain explicit information about the number of data records and the storage format of the variables, TDA reads the data in the input file two times. In a first run, it tries to determine the number of data records and the storage requirements for each variable. Memory allocation for the data is based on this information. Each variable gets the minimum storage size to save its values as found in the input file. Floating point values are always stored in double precision, i.e. with storage size 8. Storage size for string variables always equals the (negative value of the) length of the string variable.

4. TDA ignores any information on print formats that might be present in the SPSS portable file and tries to calculate appropriate print formats based on each variable's range of values. However, TDA's default behavior in creating print formats will not always give sensible results for floating point numbers,[1] and one can therefore use the `fmt` parameter to explicitly specify a print format for floating point values.

5. TDA recognizes so-called *system missing values* that may be present in the data portion of an SPSS portable file (indicated by an asterisk). By default, these missings are substituted by the numerical value -5 (or

[1] TDA uses the maximum length of the integral part of the floating point number and then provides for 6 digits after the decimal point, based on an F format.

by blanks if it is a string variable). The missing value code for numerical variables can be changed with the `msys` parameter. Note that TDA ignores any definitions of *user-defined* missing values since these definitions to not alter the data portion of an SPSS portable file.

6. By default, TDA creates an internal data matrix, or an output file defined with the `df` parameter, for all data records found in the input file. Optionally, one can use the `noc` parameter to define an upper limit of cases.

7. The `dvar` parameter can be used to create a variable description file for the variables found in the SPSS input file. The syntax is

```
dvar (argument1,argument2, ...) = output_file_name,
```

For each variable in the SPSS input file, the output file will contain one record describing the variable according to the conventions for variable description files for TDA data archives.[2] The optional arguments are as follows:

a) The argument

```
fn = file_number,
```

can be used to specify a file number that is used in the variable description file; the default file number is 1.

b) The argument

```
p = n,
```

where `n` is a positive integer. Each string variable will then be partitioned into substrings of `n` characters length, and corresponding variable definitions will be added to the variable description file. The additional variables are of type string.

c) The argument

```
pn = n,
```

where `n` is a positive integer. Same as `p = n`, but the new variables will be defined as numerical variables.

[2] If a file with the same name already exists, it is overwritten without warning. To append the variable descriptions to the end of an already existing file, one can use `dvara` instead of `dvar`.

d) The argument

```
p(VNAME) = n,
```

where `n` is a positive integer and `VNAME` is the name of a variable. Then, if `VNAME` is the name of a string variable, this variable will be partitioned into substrings of `n` characters length, and corresponding variable definitions will be added to the variable description file. The additional variables are of type string.

e) The argument

```
pn(VNAME) = n,
```

where `n` is a positive integer. Same as `p(VNAME) = n`, but the new variables will be defined as numerical variables.

Note that the `p` and `pn` arguments are incompatible. The `p(VNAME)` and `pn(VNAME)` arguments are compatible and can be used several times.

8. The `arcd` option has the syntax

arcd(argument1,argument2) = file_name,

and can be used to create, or update, an archive description file. The optional arguments are:

zoo = file_name,

This file name is then used in the archive description file for the ZOO archive.

vdf = file_name,

This file name is then used in the archive description file for the variable description file. The file number for this file is always 999.

Note that the `arcd` option is only recognized if the `rspss` command contains both a `df` and a `dvar` parameter. In order to get valid information about the number of records in the variable description file, the name used with the `vdf` parameter should be the same as the name used for the `dvar` parameter. Also note that for the calculation of physical record lengths that is required for the archive description file, it is assumed that there is a 1-byte EOL character when running under UNIX, and a 2-byte EOL character when running under DOS and MS Windows.

Writing SPSS portable files. The command for writing SPSS portable files is `wspss` with syntax shown in Box 1. All parameters, except an output file name on the right-hand side, are optional. Without additional parameters, TDA tries to create an SPSS portable file containing all currently defined variables.

1. To select a subset of variables for the output file one can use the parameter

```
keep = varlist,
```

with `varlist` a list of variables separated by commas. Then only these variables are written into the output file. Alternatively, one can use the parameter

```
drop = varlist,
```

Then all variables except variables specified in `varlist` are used.

2. By default, cases are not sorted before writing data into the output file. As an option, one can use the parameter

```
sort = V1,V2,...,
```

to sort the data matrix first w.r.t. `V1`, then w.r.t. `V2`, and so on, in ascending order.

3. Except for string variables, to define print formats for the SPSS portable file, TDA always uses an F format. A free format is translated to `10.4`, an E format is translated to `12.4`.

2.11.2.2 SPSS Sav Files

There are two commands, `rspss1` and `wspss1`, to read and write SPSS sav files that have been created by, or can be used with, SPSS for Windows running on an Intel (reversed byte order) platform. The syntax of these commands is shown in the following box.

```
rspss1(
    noc=...,        maximum number of cases, def. all
    msys=...,       system missing value code, def. -5
    df=...,         write data directly to output file
    dvar=...,       create file with variable descriptions
) = file_name;

wspss1(
    keep=...,       keep variables
    drop=...,       drop variables
    sort=...,       sort cases
) = file_name;
```

1. Except for the file name on the right-hand side, all parameters are optional.

2. The `rspss1` command can only be used when a data matrix does not already exist. By default, the command creates a new data matrix. If the `df` parameter (or `dfa` to append data) is used, the data are written directly to an output file and a data matrix is not created.

3. The `dvar` parameter can be used to request an additional output file that contains a description of variables (variable labels and, if present, also value labels) as found in the input file. Note however, that this will not be a variable description file in the sense of TDA's data archives.

4. The `rspss1` command assumes that the SPSS sav file was created by SPSS for Windows on an Intel platform, meaning that binary values are stored in reversed byte order. Correspondingly, the binary files

created by the `wspss1` file are only appropriate for this architecture.

5. The `keep`, `drop`, and `sort` parameters in the `wspss1` command expect comma-separated lists of variables. String variables cannot be used for sorting.

2.11.3 Stata Files

There are two commands to read and write Stata files (versions 4.0 and 6.0). Implementation is based on the description of the `.dta` file format in the documentations of Stata releases 4.0 and 6.0.

Reading Stata files. The command for reading Stata files is `rstata` with syntax shown in Box 1. On the right-hand side the name of a Stata data file must be given. All other parameters are optional. By default, TDA tries to create a data matrix based on the information in the input file. Alternatively, the `df` parameter can be used to define the name of an output file. The data read from the Stata file are then written directly into that output file without creating an internal data matrix. Note that in both cases the command is not executed if a data matrix already exists.

1. When reading Stata files, TDA should be able to correctly recognize whether the data are stored according to a HiLo or LoHi byte ordering.

2. If the first character of a Stata variable name does not conform to the TDA conventions for variable names (see 2.1), it is preceded by an underline character, otherwise it is always converted to an upper case letter.

3. Translation of Stata variable types to TDA variables types is as follows:

1. Stata variables of type *b* (byte) are stored with storage size 1.
2. Stata variables of type *i* (short integer) are stored with storage size 2.
3. Stata variables of type *l* (long integer) are stored with storage size 5.
4. Stata variables of type *f* (single precision floating point) are stored with storage size 4.
5. Stata variables of type *d* (double precision floating point) are stored with storage size 8.

Box 1 Syntax of `rstata` and `wstata` commands

```
rstata (
    noc=...,          maximum number of cases, def. all
    msys=...,         system missing value code, def. -5
    df=...,           write data directly to output file
    dvar()=...,       create/update variable description file
    arcd()=...,       create/update archive description file
) = fname;

wstata (
    keep=...,         keep variables
    drop=...,         drop variables
    sort=...,         sort cases
    ptyp=...,         type of output file, def. 6
                      4 : for Stata release 4
                      6 : for Stata release 6
) = fname;
```

6. Stata string variables will result in corresponding TDA string variables.

4. The print format for numerical variables is based on the information in the Stata file. Note, however, that *g*-type formats are translated to *f*-type formats.

5. A Stata input file may contain missing value codes. These are substituted by the numerical value -5. Another missing value code can be specified with the `msys` parameter.

6. By default, TDA creates an internal data matrix for all data records found in the input file or. Using the `df` parameter, all records are written into the specified output file without creating an internal data matrix. Optionally, one can use the `noc` parameter to define an upper limit of cases.

7. The `dvar` parameter can be used to create a variable description file for the variables found in the Stata input file. The syntax is

```
dvar (optional arguments) = output_file_name,
```

For a description of the optional arguments see **2.11.2.1**. Also note that

one can use `dvara` to append information to an already existing file. Note that the `rstata` command only tries to find value label information if the `noc` parameter is not used. This is due to the fact that, in a Stata file, value label information is always at the end of the file.

8. The `arcd` parameter can be used to create an archive description file. For an explanation of this option see **2.11.2.1**.

Writing Stata files. The command for writing Stata files is `wstata` with syntax shown in Box 1. All parameters, except an output file name on the right-hand side, are optional. Without additional parameters, TDA tries to create a Stata file containing all currently defined variables.

1. By default, the `wstata` command creates a Stata file for release 6.0. The `ptyp` parameter can be used to create a file for release 4.0.

2. To select a subset of variables for the output file one can use the parameter

   ```
   keep = varlist,
   ```

   with `varlist` a list of variables separated by commas. Then only these variables are written into the output file. Alternatively, one can use the parameter

   ```
   drop = varlist,
   ```

   Then all variables, except those specified in `varlist`, are used.

3. By default, cases are not sorted before writing data into the output file. One can use the parameter

   ```
   sort = varlist,
   ```

   to sort the data matrix, based on the variables contained in `varlist` (from left to right), in ascending order. Note that string variables cannot be used for sorting.

2.12 Data File Utilities

This chapter describes commands which are sometimes helpful to investigate and manipulate (data) files. The sections are as follows.

2.12.1 Characters in a File explains the `ccnt` command that can be used to get a frequency distribution of the characters in a file.

2.12.2 Records in a File explains the `lcnt` command that can be used to get a frequency distribution of the lengths of records in a file.

2.12.3 Binary Contents of a File explains the `dump` command that can be used to see into the binary contents of a file.

2.12.4 Splitting Files into Parts explains the `dsplit` command that can be used to split a file into parts.

2.12.5 External Sorting explains the `esort` command that can be used to sort big data files.

2.12.6 External Merging explains the `emerge` command that can be used to merge big data files.

2.12.7 Selection of Records explains the `eselect` command that can be used to select records from big data files.

2.12.8 Dropping Selected Columns explains the `eskip` command that can be used to drop selected columns from big data files.

2.12.1 Characters in a File

The `ccnt` command can be used to get a frequency distribution of the characters contained in a file. The syntax is

```
ccnt = fname;
```

where `fname` is the name of a file. The resulting frequency distribution is written into the standard output. The characters in the file are shown in their hexadecimal representation and, if possible, as visible ASCII characters.

2.12.2 Records in a File

The `lcnt` command can be used to get a frequency distribution of the lengths of records in a file. The syntax is

```
lcnt (noc=...) = fname;
```

where `fname` is the name of a data file. It is assumed that this file has end-of-record characters: single line-feed characters (UNIX) or carriage-return line-feed sequences (DOS and DOS-like platforms). The end-of-record characters are *not* included in counting the record length.

The `noc` parameter is optional and can be used to set an upper limit for the number of cases, default is `noc=1000`. The maximum record length is 20000 characters.

2.12.3 Binary Contents of a File

The `dump` command can be used to show the binary contents of a file. Each character is shown in its hexadecimal representation and, if possible, as a visible ASCII character. The syntax is

```
dump (nc=...,s=...) = fname;
```

where `fname` is the name of a file. Parameters are optional. By default, the command begins with the first character in the input file (`s = 0`). Alternatively, it begins with an offset given by the `s` parameter. The `nc` parameter can be used to limit the number of characters; by default, all characters of the input file are used.

2.12.4 Splitting Files into Parts

The `dsplit` command can be used to split a file into parts. The syntax of this command is

```
dsplit (len=...) = fname;
```

where `fname` is the name of a file. The size of the parts can be defined with the parameter `len`, default is `len=1000`. The file specified by `fname` is then split into parts of `n` bytes (of course, the last part could be less than `n` bytes if the size of `fname` is not a multiple of `n`). The parts are named:

`fname.a`, `fname.b`, `fname.c`, and so on.

Note that on DOS-like platforms, `fname` should not already contain a period (.) because file names may only contain a single point.

2.12.5 External Sorting

The `esort` command can be used to sort big data files that do not fit into main memory. The syntax is shown in the following box.

```
esort (
    df=...,          name of output file (required)
    sk=...,          up to five sort keys
    noc=...,         number of records in temp. file, def. 1000
    len=...,         maximal record length, def. 1000
) = input_file;
```

The command expects the name of a standard ASCII data file on its right-hand side. This file must have valid end-of-record characters. The maximal record length can be specified with the `len` parameter. Also required is the name of an output file to be given with the `df` parameter. Finally, at least one sort key must be defined with the `sk` parameter. The syntax is

$$\texttt{sk} = i_1, j_1, i_2, j_2, \ldots,$$

where the right-hand side is a sequence of up to five pairs of column numbers. Each pair specifies one sort key. The first sort key is taken to be the field from column i_1 up to, and including, column j_1; and so on. If there are two or more sort keys they are used hierarchically from left to right.

The command works as follows.

1. The command reads up to `noc` records from the input file and creates a corresponding array of sort keys and pointers to the data file records. This array is then sorted in ascending order and the sorted records are written into a temporary output file.

2. Step 1 is repeated until no more records can be read from the input file.

3. Finally, the temporarary files are simultaneously read and merged in order to create a new sorted output file.

One should note that the maximal number of temporary files is 100, but the actually available number may be less due to limitations of the operating system. One should also note that names of temporary files are created by adding ".i" to the name of the input file ($i = 0, 1, 2, \ldots$). Therefore, when working on a DOS-like platform, the name of the input file should not already contain a period.

2.12.6 External Merging

The `emerge` command can be used to merge big data files that do not fit into main memory. The syntax is shown in the following box.

```
emerge (
    df=...,          name of output file (required)
    mf=...,          list of merge files
    len=...,         maximal record length, def. 1000
    m=...,           missing value character, def. blank
    sepc=...,        separation character, def. blank
) = input_file;
```

The command expects the name of a standard ASCII data file on its right-hand side. This file must have valid end-of-record characters. The maximal record length can be specified with the `len` parameter. Also required is the name of an output file to be given with the `df` parameter. Finally, at least one more file to be merged with the input file must be specified with the `mf` parameter. The syntax is

$$\begin{aligned} \texttt{mf} &= \text{fname1}\,[i_1, j_1, i, j], \\ &= \text{fname2}\,[i_2, j_2, i, j], \\ &= \text{fname3}\,[i_3, j_3, i, j], \cdots, \end{aligned}$$

Up to 50 file names can be specified on the right-hand side. (Note, however, that this maximum may be less due to limitations of the operating system.) Like the main input file, these files must be plain ASCII files with valid end-of-record characters and record length not exceeding `len`. In addition, for each of these files one has to specify two fields. The first two numbers, i_k and j_k, must specify, respectively, the first and last columns of a field in the records of the corresponding file to be merged (i.e., `fname`k). The second two numbers, i and j, must specify a field in the main input file. While the (i_k, j_k) fields can be different for each file, there can only be a single field in the main input file. Furthermore, all fields must have the same length.

The command expects that all input files are already sorted with

respect to the specified fields. In fact, the command sequentially reads records from the main input file and, at the same time, reads records from all other input files. In case of identical fields records from the additional input files are added to the main record, otherwise it is filled with missing value characters.

By default, missing records are substituted by blank characters. Optionally, one can use the `m` parameter to specify one of the digits $(0, 1, \ldots, 9)$. Also by default, records from the input files are separated by a single blank character. Optionally, one can specifiy an alternative separation character with the `sepc` parameter, or suppress any separation character with "`sepc=none`".

2.12.7 Selection of Records

The `eselect` command can be used to select records from a big data file based on keys provided by another file. The syntax is shown in the following box.

```
eselect (
    df=...,          name of output file (required)
    if=...,          additional input file
    sk=...,          specification of keys
    noc=...,         max number of keys, def. 1000
    opt=...,         if 1 (def.) write all records
                     if 2 supress identical records
    len=...,         maximal record length, def. 1000
) = input_file;
```

The command expects the name of a standard ASCII data file on its right-hand side. This file must have valid end-of-record characters. The maximal record length can be specified with the `len` parameter. Also required is the name of an output file to be given with the `df` parameter.

In addition, the command expects another input file to be specified with the `if` parameter. Again, this must be a standard ASCII file with valid end-of-record characters and record length not exceeding `len`. The parameter

$$\texttt{sk} = i_1, i_2, j_1, j_2$$

must be used to specify two fields. i_1 and i_2 specify, respectively, a field beginning in column i_1 and ending in column i_2 in the records of the main input file. j_1 and j_2 specify a corresponding field in the records of the additional input file defined with the `if` parameter. Both fields must have the same length. The maximal number of fields (keys) read from the additional input file can be specified with the `noc` paramater.

The command works as follows. It reads up to `noc` keys from the additional input file defined with the `if` parameter. These keys are then sorted. Finally, the command reads the records from the main input file

and whenever a matching key is found, this record is written to the output file. By default, all matching records are written into the output file. If the `opt=2` parameter is given each key is used only once.

2.12.8 Dropping Selected Columns

The `eskip` command can be used to drop selected columns from a big file. The syntax is shown in the following box.

```
eselect(
    df=...,          name of output file (required)
    sk=...,          up to five fields
    len=...,         maximal record length, def. 1000
) = input_file;
```

The command expects the name of a standard ASCII data file on its right-hand side. This file must have valid end-of-record characters. The maximal record length can be specified with the `len` parameter. Also required is the name of an output file to be given with the `df` parameter.

In addition, the command expects a specification of up to five fields with the parameter

$$\texttt{sk} = i_1, j_1, i_2, j_2, \ldots,$$

The kth field begins in column i_k and ends in column j_k. The command sequentially reads the records from the input file and write those columns into the output file that do not fall in any of the column ranges defined with the `sk` parameter.

3. Data Structures

TDA supports several different data structures, each based on a specific interpretation of the internal data matrix. Some of these data structures must be explicitly specified in order to allow subsequent commands to recognize the data structure. This part of the manual describes which data structures are available and, if necessary, explains commands that must be used to specify these data structures.

3.1 Cross-sectional Data

We shall use the term *cross-sectional data* in a broad sense, meaning a data structure that simply consists of a set of variables without explicitly recognizing any additional hierarchical or temporal structure. Thus, a cross-sectional data structure directly corresponds to a rectangular data matrix,

Case	X_1	X_2	$\cdots$	X_m
1	x_{11}	x_{12}		x_{1m}
2	x_{21}	x_{22}		x_{2m}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
n	x_{n1}	x_{n2}		x_{nm}

containing m variables for n cases. Therefore, in order to interpret TDA's internal matrix as a set of cross-sectional data, it is not necessary to use any further specification. One can directly refer to the variables in the internal data matrix.

3.2 Panel Data

We speak of *panel data* if we have a set of variables for several points in time. A variable may then be denoted by X_{jt} where $j = 1, \ldots, m$ identifies the variable and $t = 1, \ldots, T$ the point in time. (Of course, instead of thinking of different points in time, the index t can refer to any other kind of grouping.) The value of X_{jt} for the ith individual will be denoted by x_{ijt}.

There are two possibilities to interpret a rectangular data matrix as a set of panel data. The data for each individual can be given in a single row, or in a sequence of T rows.

Horizontal Data Organization. In this case we assume that there is a single row for each individual. The data structures looks as follows.

Case	X_{11}	$\cdots$	X_{m1}	$\cdots$	X_{1T}	$\cdots$	X_{mT}
1	x_{111}		x_{1m1}	$\cdots$	x_{11T}		x_{1mT}
2	x_{211}		x_{2m1}	$\cdots$	x_{21T}		x_{2mT}
$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$
n	x_{n11}		x_{nm1}	$\cdots$	x_{n1T}		x_{nmT}

This is TDA's default data structure for panel data. It has the advantage that variables that do not vary across time need only be stored once.

Vertical Data Organization. Alternatively, one can assume that the data for each individual are given in a block of T data matrix rows. The data structure is then:

Case	Wave	X_1	X_2	$\cdots$	X_m
1	1	x_{111}	x_{121}		x_{1m1}
1	2	x_{112}	x_{122}		x_{1m2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
1	T	x_{11T}	x_{12T}		x_{1mT}
2	1	x_{211}	x_{221}		x_{2m1}
2	2	x_{212}	x_{222}		x_{2m2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
2	T	x_{21T}	x_{22T}		x_{2mT}
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
n	1	x_{n11}	x_{n21}		x_{nm1}
n	2	x_{n12}	x_{n22}		x_{nm2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
n	T	x_{n1T}	x_{n2T}		x_{nmT}

If the data structure is balanced, meaning that we have data (possibly missing values) for all individuals for each point in time, it should be possible to create a vertically organized data structure from a horizontal data structure, and vice versa, by using the `nc` and `nq` options in TDA's `pdata` command, see 2.9.

3.3 Episode Data

The basic concepts for defining event-history data are a time axis, $\mathcal{T}$, and a discrete state space, $\mathcal{Y}$. One possibility to represent event-history data is by an ordered set of state variables: $Y(t)$ $(t \in \mathcal{T})$. If the time axis is discrete, we can simply think of a sequence of state variables, $Y(t)$, for $t = 0, 1, 2, 3, \ldots$ This representation of event-history data is called *sequence data* and will be further discussed in **3.4**. An alternative way to represent event-history data uses the concept of *episodes*, also called *spells*, and we then speak of *episode* or *spell data*. This chapter introduces the concept of episode data as used in TDA. The subsections are as follows.

3.3.1 Episode Data Concepts

An episode, or spell, is the duration an individual stays in a specific state. The episode begins at the time of entry into that state, and it ends when a new state is entered. Of course, the definition of episodes depends on what is regarded as states; and this, in turn, depends on the substantive issue in question. As implied by the term event-history data, we always assume a discrete state space.

We speak of episode data if there is a sample of $i = 1, \ldots, N$ episodes. Thinking in terms of individuals, there might be just one episode for each individual, or each individual might contribute a varying number of episodes to the data set. Accordingly, one sometimes distinguishes between single and multi-episode data. Of course, also multi-episode data where each individual contributes a varying number episodes can be viewed as just a sample of individual episodes. This view will be taken until the end of this section where a few more remarks about multi-episode data will be given.

Each of the episodes can be described formally by an expression like

$$(o_i, d_i, s_i, t_i, x_i(t)) \qquad i = 1, \ldots, N \tag{1}$$

s_i and t_i are the *starting* and *ending times*, respectively. Since measurement is always discrete one has to adopt a convention about coding these variables. *The basic convention for TDA is that starting and ending times are coded such that $t_i - s_i$ is the duration of the episode.*

o_i is the *origin state*, the state held during the episode until the ending time, and d_i is the *destination state* defined as the state reached at the ending time of the episode. $x_i(t)$ is a vector of covariates connected with the episode and possibly depending on the process time, t. To simplify notation, we generally use the convention that covariates are given by row vectors and parameters (coefficients) are given by column vectors.

One more piece of information could be added to the description of episodes given in (1): case weights. It would be straightforward, then, to use such weights in all formulas based on a sample of episode data. To simplify the notation we will not write the formulas in this text with weights. However, if case weights have been defined with the `cwt` command, these case weights will be used in all methods based on episode data.

In all applications the sets of origin and destination states are finite. It is assumed throughout this text that they are coded by non-negative integers, including zero. The set of possible origin states is called $\mathcal{O}$. Then, for each $j \in \mathcal{O}$, there is a set of different destination states called $\mathcal{D}_j$. Accordingly, with this terminology, it is assumed that $k \neq j$ if $j \in \mathcal{O}$ and $k \in \mathcal{D}_j$. Clearly, if an episode is right censored there is no transition to a destination state different from the origin state. The set of all possible destination states, for episodes with origin $j \in \mathcal{O}$, ending with an event or are censored, is $\mathcal{D}_j \cup \{j\}$.

In addition, we generally use the following notation. Each pair (j, k) with $j \in \mathcal{O}$ and $k \in \mathcal{D}_j$ is called a *transition*. $\mathcal{N}_j$ is the set of all episodes with origin state j; $\mathcal{E}_{jk}$ is the set of all episodes with a transition from origin state j to a different destination state $k \in \mathcal{D}_j$; $\mathcal{Z}_j$ is the set of all censored episodes with origin state j; and $\mathcal{Z}_{jk}$ is the set of all episodes with origin state j and with a destination state not equal to k.

The Time Axis. As with any longitudinal data it is important how the time axis is defined and the relevant dates are measured. We distinguish three different aspects of this problem.

• One needs a time axis that is suitable for the substantive application. This, of course, depends on the kind of events that one intends to model. In sociological applications, this will normally be a discrete time axis, for instance days. The important point is that all events that are normally investigated in sociological research have some inherent duration, for instance a birth or becoming married.

• In order to formulate a formal (statistical) model we then need a mathematical representation of time. Here we have two choices; we can use a discrete or a continuous time axis. In our view, there are no principal arguments to prefer one over the other. It is often convenient to use a continuous time axis, and in fact, most transition rate models discussed in **6.17** use a continuous time axis to mathematically represent the substantive process.

• It remains the question of how dates of events have been measured and are available in a given set of episode data. This information is, of course, always given in discrete time units. For instance, in sociological and demographic research, dates are normally given by the month when an event happened. The problem then is how to relate these empirical dates to the mathematical time axis that has been used to define the model.

No specific problem arises if the mathematical time axis is discrete with the same time units as used for the empirical dates. Otherwise, there are two different possibilities. One can simply assume that the empirical dates can be exactly mapped to the mathematical time axis; or one can represent the empirical dates by time intervals on the mathematical time axis. Current applications of event-history methods almost always use the method of simply identifying empirical dates with some time points on the mathematical time axis.

If empirical dates are measured very imprecisely, the second approach, based on mapping these dates to time intervals on the mathematical time axis, would be preferable. This would then appropriately reflect the fact that if dates are measured very imprecisely, we only know that the event happened in some time interval. This is sometimes called *interval censored dates*. Of course, there is no clear-cut demarcation line between precisely measured and interval censored dates. In any case, statistical methods for interval censored data are currently not supported by TDA.

Right Censored Data. If event histories could be observed completely, each individual will eventually reach an absorbing state. However, each sample of episode data is based on a limited observation period. Most probably, it will contain some episodes without an observed transition to a new destination state. These episodes are called *right censored*.

Since almost all episode data sets contain at least some right censored episodes, this must be taken into account by all methods appropriate for such data. In order to do this one needs some information about the process that generated the censoring. One can distinguish, at least, three different types of censoring (Lawless [1982, p. 31]).

- Type I Censoring. This is a special type of censoring arising in experiments where the test is for the duration in some state. When there is a fixed time period during which a set of individuals, or other units of analysis, is exposed to some risk of leaving the initial state, and if this fixed time period is the same for all units, then the resulting censoring is called type I censoring.

- Type II Censoring. This type of censoring again arises in experiments where the test is for the duration in some state. But now there is not a fixed time period for observation, but the experiment is stopped after an event has occurred for a fixed fraction of all units.

- Random Censoring. This third type of censoring is more general and

not restricted to experiments. It is used to describe situations where the finite observation period for a sample of episode data, which generates the resulting censoring, is statistically independent from the durations that are to be studied.

In sociological research, based mainly on survey data and follow-up studies, random censoring can be assumed in most cases. Fortunately, in practice, it is not necessary to deal separately with all different types of censoring. There is only one really important assumption: that the process generating the censoring works independently from the process that generates the transitions one wishes to investigate. All methods to analyze episode data, currently implemented in TDA, are based on this assumption.

Left Censored and Truncated Data. An Episode is called *left censored* if we do not know its starting time. This is a more complicated situation because there is simply no substitute for the missing information. Sometimes it will be possible to assume a time interval for the starting time of a left censored episode. If this would be possible, left censored episodes could be treated as a special case of interval censored episodes. But this is currently not supported by TDA and we consequently assume that there is an approximately exact starting time for all episodes to be used with TDA.

A quite different situation arises if the observation of episodes is *left truncated*, meaning that the episode is not observed from its beginning, but we are able to find out its starting time retrospectively. Most statistical methods offered by TDA can also be used with left truncated data as will be explained when describing the methods.

Basic Statistical Concepts. We now introduce some statistical concepts for representing episode data defined on a continuous time axis. Analogous concepts for a discrete time axis will be defined in 6.17.6.1.

Assuming that the episodes are defined on a process time axis where each episode begins at time zero, and assuming that there is only a single destination state, a sample of episodes can be completely represented by a non-negative stochastic variable T. The distribution of such a variable can be described by a density function $f(t)$, or by a distribution function $F(t)$, with the simple relation

$$\Pr(T \leq t) = F(t) = \int_0^t f(\tau)\, d\tau$$

The use of well-behaved distributions, i.e. $F(t) \to 1$ for $t \to \infty$, implies that all episodes end at some point in time with an event. But obviously, this is not always true. For instance, if we are interested in the distribution of durations, starting with birth and ending with first marriage, then one has to provide for the possibility that some people will never get married. An appropriate way to model such situations is to allow for two or more alternative destination states. In this example, one possible destination state is to become married; and an alternative destination state is to die before becoming married.[1]

Two more concepts are often used. First the concept of a *survivor function*, generally denoted by $G(t)$. It gives the probability of not to have an event until the point in time t, so the definition is

$$G(t) = \Pr(T > t) = 1 - F(t)$$

Another important concept, called the *hazard* or *transition rate*, is defined by

$$r(t) = \lim_{\Delta t \to 0} \frac{\Pr(t \leq T < t + \Delta t \mid T \geq t)}{\Delta t} \tag{2}$$

The numerator of this expression is the conditional probability of having an event in a small time interval from t to $t + \Delta t$, conditional on having no event until t. Then this conditional probability is measured per unit of time and the limit is taken. Consequently, the transition rate describes the *instantaneous rate*, or sometimes called the *risk*, of having an event at time t.

As is easily seen, the transition rate can be expressed by the density and the survivor function of the duration variable T.

$$r(t) = \frac{f(t)}{G(t)} \tag{3}$$

Therefore, all four concepts are mathematically equivalent. The density function, the distribution function, the survivor function, and the transition rate can equally well be used to describe the duration of episodes.

[1] Another approach is to use so-called *mover-stayer models*, based on the assumption that there is an unknown, but estimable proportion of individuals who will never experience a transition into the destination state. Whether this is a sensible assumption depends on the application. For instance in life course research, the assumption conflicts with the basic view that life courses are contingent developmental processes. In our example, this would mean that the risk of becoming married remains until death.

Furthermore, from (3) one gets the important equation

$$G(t) = \exp(-H(t)) = \exp\left\{-\int_0^t r(\tau)\,d\tau\right\} \tag{4}$$

with

$$H(t) = \int_0^t r(\tau)\,d\tau$$

called the cumulative hazard or transition rate.

Alternative Destination States. So far we only considered transitions from a single origin state into a single destination state. Duration was defined as staying in the origin state until a transition to the destination state occurred, and if none occurred during the observation period, the episode was regarded as right censored. However, there may be more than one transition of interest represented in a sample of transition data. There may be more than one origin state, and, for each origin state, there may be more than one destination state.

The possibility of more than one origin state can be dealt with by conditioning all descriptions and modeling on a given origin state. This is the standard approach: all analyses are done conditional on being in a given origin state. For instance, one may be interested in the duration of poverty spells; descriptions of these durations are then given only for persons who entered into a state of poverty.

Somewhat more involved is the treatment of situations where a given origin state can be followed by one of several possible destination states. We then have to treat the destination state variable as stochastic. Conditional on a given origin state j, we can conceptualize such a situation by a two-dimensional stochastic variable (T_j, D_j). The variable T_j gives the duration in the origin state j until *any* event occurs, or until the episode becomes right censored, and $D_j \in \mathcal{O}_j \cup \{j\}$ gives the destination state reached at T_j, i.e. at the end of the episode, conditional on the origin state j.

The goal then is to describe this two-dimensional variable.[2] We can first look at the marginal duration variable T_j, i.e. the duration in the

[2]Cf. for the following concepts Blossfeld et al. [1986, p. 59], and Lancaster [1990, p. 99]. A discussion of the competing risks problem in the biometrical context is given by Gail [1975]. However, we should clearly distinguish between the biometrical concept of competing risks and situations with alternative destination states. In the classical competing risks case, there is only a single destination state (death of an individual,

origin state j until any event occurs. The distribution of this variable may be described by overall survivor and density functions, and by an overall transition rate

$$G_j(t) = \Pr(T_j > t)$$

$$f_j(t) = -\frac{d}{dt} G_j(t)$$

$$r_j(t) = \frac{f_j(t)}{G_j(t)}$$

With these concepts we get a description for the duration as if we had collapsed all different destination states into only one. But, in fact, we are interested in the different risks associated with the different destination states. So the concepts must be distinguished accordingly.

This is most easily done using the transition rate.[3] In generalizing (2) the transition rate from the given origin state j to destination state $k \in \mathcal{D}_j$ is defined as

$$r_{jk}(t) = \lim_{\Delta t \to 0} \frac{\Pr(t \le T_j < t + \Delta t, D_j = k \mid T_j \ge t)}{\Delta t} \tag{5}$$

This gives the risk, or the conditional probability if we look at a small time interval Δt, that a transition from the given origin state j to the destination state k takes place at t, or in the time interval $[t, t + \Delta t)$, given that no event occurred until time t.

Because we always assume that the destination states are mutually exclusive, the overall transition rate $r_j(t)$, from the given origin state j to any destination state, can be expressed as

$$r_j(t) = \sum_{k \in \mathcal{D}_j} r_{jk}(t)$$

Using the general equality (4) it follows that

$$G_j(t) = \exp\left\{-\int_0^t r_j(\tau)\, d\tau\right\} = \prod_{k \in \mathcal{D}_j} \exp\left\{-\int_0^t r_{jk}(\tau)\, d\tau\right\}$$

---

or failure of a machine), but two or more risk factors can lead to this event. In typical sociological applications the situation is quite different: individuals can leave a given origin state into one of several possible destination states.

[3] In the case of two or more destination states, the transition or hazard rate is often called a *transition intensity*. However, it should not cause undue confusion to use the same term *transition rate* in all cases.

This shows that the overall survivor function $G_j(t)$ may be written as a product of *pseudosurvivor functions* defined as

$$\tilde{G}_{jk}(t) = \exp\left\{-\int_0^t r_{jk}(\tau)\, d\tau\right\}$$

They are called this way because there is no direct survivor function interpretation of these functions. They are, however, quite useful when deriving likelihood expressions for transition rate models.

To get concepts for a direct interpretation of destination-specific transition rates we continue with a definition of transition-specific distribution functions $\tilde{F}_{jk}(t)$, giving the probability that a transition to destination state k, from the given origin state j, takes place until t.

$$\tilde{F}_{jk}(t) = \Pr\left(T_j \leq t, D_j = k\right) \tag{6}$$

Although the meaning of this concept is obvious it is not a proper distribution function in the normal sense, since we have

$$\tilde{F}_{jk}(\infty) = \pi_{jk}$$

where π_{jk} is the probability of reaching, at any point of time, the destination state k, given the origin state j; and if there is more than one possible destination state this probability is clearly less then one. Therefore, (6) is sometimes called a *subdistribution function*, e.g. by Kalbfleisch and Prentice [1980, p. 167]. The same is to be said if we look at the density functions

$$\tilde{f}_{jk}(t) = \frac{d}{dt}\tilde{F}_{jk}(t)$$

This, again, is not a proper density function, because it does not integrate to unity, and so it is sometimes called a *subdensity function*. However, we have

$$\tilde{f}_{jk}(t) = \lim_{\Delta t \to 0} \frac{\Pr\left(t \leq T_j < t + \Delta t, D_j = k\right)}{\Delta t} \tag{7}$$

and so it is easy to adjust our definitions to become

$$F_{jk}(t) = \frac{\tilde{F}_{jk}(t)}{\pi_{jk}} \tag{8}$$

$$f_{jk}(t) = \frac{\tilde{f}_{jk}(t)}{\pi_{jk}} \tag{9}$$

Now $F_{jk}(t)$ and $f_{jk}(t)$ may be regarded as concepts describing distributions in the usual sense.[4] The range of this distributions may be thought of as the set of all individuals who leave the given origin state for a specific destination state. And the overall distribution $F_j(t)$ may be regarded as a mixture of these destination-specific distributions, with the π_{jk} as weights:

$$F_j(t) = \sum_{k \in \mathcal{D}_j} \tilde{F}_{jk}(t) = \sum_{k \in \mathcal{D}_j} \pi_{jk} \, F_{jk}(t) \tag{10}$$

It seems not sensible to define destination-specific survivor functions since this would imply a "conditioning on the future". It is quite possible, however, to express destination-specific transition rates by the overall survivor function and destination-specific densities:

$$r_{jk}(t) = \frac{\tilde{f}_{jk}(t)}{G_j(t)} = \pi_{jk} \, \frac{f_{jk}(t)}{G_j(t)} \tag{11}$$

The first part follows from (5) and (7), the second part follows from (9).

The focus of modeling can finally be described as follows: Estimation and evaluation of models for stochastic variables (T_j, D_j) that measure the duration of being in an origin state j until a transition to a destination state D_j occurs. This is done conditionally on a given origin state.

Multi-episode data. The term *multi-episode data* is normally used to denote a situation where each individual contributes a varying number of episodes to a given episode data set. There is then a set, $\mathcal{U}$, of individuals, and for each individual $u \in \mathcal{U}$, there is a series of M_u episodes ($M_u \geq 1$). This results in a set of N episodes,

$$N = \sum_{u \in \mathcal{U}} M_u$$

A formal representation can be given similar to the single episode case. One only has to add, for each single episode, two more pieces of information: an identification number of the individual to which the episode belongs, and the serial number of the episode. A complete description of a sample of multi-episode data is then given by

$$(u_i, m_i, o_i, d_i, s_i, t_i, x_i) \qquad i = 1, \ldots, N \tag{12}$$

[4] We assume that the probabilities π_{jk} add to unity. This assumption can be met by defining a complete destination state space.

where u_i is the identification number of the individual, or any other unit of analysis, the ith episode belongs to; and m_i is the serial number of the episode. All other variables are defined as in the single episode case. o_i is the origin state, d_i is the destination state, s_i is the starting time, t_i is the ending time, and x_i is a vector of covariates associated with the episode.

Compared with the single episode case, there are now a few more consistency requirements. It is not only required that the duration of each episode is greater than zero. In addition, there is an ordering of the episodes for each individual, given by the set of serial numbers for the episodes. In particular, it is required that the starting time of an episode is not less than the ending time of a previous episode.

As already mentioned, a set of multi-episode data can also be viewed as a set of single episodes, simply disregarding the connection between episodes and individuals. Of course, statistical models for episode data should then take into account that episodes contributed by the same individual are probably not independent. However, this requirement is conditional on covariates and it is possible, therefore, to model multi-episode data by adding, for each episode, covariates representing relevant aspects of its past history.

While TDA is able to recognize the distinction between single and multi-episode data, transition rate modeling is normally not done with multi-episode data. Instead, one uses the just mentioned approach. The data set is treated as a set of single episodes, and covariates are added to control for serial dependence across time. In fact, the concept of multi-episode data is only necessary when the analytical focus is not on transitions, but on describing individual event histories over a longer period of time. However, episode data structures are not well suited for this purpose. The concept of sequence data provides a useful alternative, see **3.4**.

3.3.2 Defining Episode Data

The basic command for defining an episode data structure is `edef` with syntax shown in Box 1. The first four parameters are required.[1] The parameter

```
org = expression,
```

must be used to define an origin state for the episodes. `expression` can be a numerical constant, for instance, `org=0` would define a zero origin state for all episodes; or can refer to a data matrix variable, for instance `org=ORG` would define the origin state by referring to variable `ORG`. Also more complicated expression are possible but, in general, it is more efficient to define variables outside the `edef` command. In the same way, one has to use

```
des = expression,
```

to define a destination state,

```
ts = expression,
```

to define a starting time, and

```
tf = expression,
```

to define the ending time of the episodes. Note that origin and destination states must be nonnegative integers; also all starting times must be nonnegative, and each episode must have a strictly positive duration, always calculated as `tf - ts`.

Given these parameters in the `edef` command, TDA creates an episode data structure based on the currently selected cases of the data matrix. This data structure remains active until it is replaced by a new `edef` command, the `tsel` command is used for a new temporary case selection, or the `clear` command is used to remove a variable required to maintain the episode data structure. If TDA has successfully created an episode

[1] If the `edef` command is used without any parameters it provides information about the currently defined episode data (if any).

Box 1 Syntax for `edef` command

```
edef (
      org=...,          origin state
      des=...,          destination state
      ts=...,           starting time
      tf=...,           ending time
      id=...,           ID number
      sn=...,           spell number
      maxtran=...,      maximum number of transitions, def. 100
      VName=...,        definition of type 5 variables
      ...               can be repeated
      split=...,        variables for episode splitting
);
```

data structure, it shows a table with information about all transitions in the standard output.

Except for `org`, `des`, `ts` and `tf`, all other parameters in the `edef` command are optional.

1. The `maxtran` parameter can be used to change the maximum number of transitions, default is `maxtran=100`.

2. The `id` and `sn` parameters can be used to define multi-episode data. The syntax is

```
id = expression,
```

to define an ID number for the episodes, and

```
sn = expression,
```

to define spell numbers. TDA then assumes that the episodes in the data matrix are ordered with respect to the ID variable. Spell numbers must be nonnegative integers and should be in ascending order.

3. The `split` parameter can be used to define variables for episode splitting. If used, it must be the last parameter in the `edef` command. The syntax is

```
split = varlist,
```

where `varlist` is a list of variables separated by commas. Type 5 variables (see below) cannot be used for episode splitting. Splitting is done as

follows. Let (o_i, d_i, s_i, t_i) denote the ith episode and S_i the corresponding value of one of the variables specified in `varlist`. Then, if $s_i < S_i < t_i$, the episode is split into two parts: (o_i, o_i, s_i, S_i) and (o_i, d_i, S_i, t_i). This is done recursively for all variables in `varlist`. Note that episode splitting does not change the internal data matrix. Episode splitting is only performed when (and while) procedures request episode data. In fact, currently only three commands recognize episode splitting: the `epdat` command for writing episode data into an output file and the `rate` and `frml` commands for estimating transition rate models. If the `split` parameter was used in the `edef` command, these procedures simply get splits whenever they request a new episode.

4. The `edef` command can also be used to create temporary (type 5) variables based on the episode data structure. The syntax is

```
VName <s>[pfmt] = expression,
```

and is identical to the syntax used for defining variables with the `nvar` command (see 2.2). `VName` is the variable name, `<s>` is an optional storage size, default is 4, and `[pfmt]` is an optional print format. In addition to all standard operators, `expression` may also contain the episode data operators, `org`, `des`, `ts`, `tf`, and `sn`, described in 5.2.8. These operators provide the corresponding values for the current episode, or split, and can be used to create time-varying variables based on episode splitting.

3.3.3 An Example Data Set

To illustrate episode data we use an example data set taken from the German Life History Study (GLHS) collected at the Max Planck Institut für Bildungsforschung in Berlin. The same data set is used in Blossfeld and Rohwer [1995]. We thank Karl Ulrich Mayer and Hans Peter Blossfeld who kindly provided this data set.

The GLHS provides retrospective information about the life histories of men and women from the birth cohorts 1929–31, 1939–41, and 1949–51, collected in the years 1981–1983 (Mayer and Brückner, 1989). Our example data set contains 600 job episodes from 201 randomly selected respondents. Each record in this file represents an employment episode, and the consecutive jobs of a respondent's career are stored successively in the file. For some individuals there is only a single job episode, whereas for others there is a sequence of two or more jobs.

The data file, `rrdat.1`, contains 12 variables that are described briefly in Box 2. `Column` refers to the position of the variable in the data file, which is free format, meaning that the numerical entries are separated by a blank character.

`ID` identifies the individuals in the data set. Because the data file contains information about 201 individuals, there are 201 different ID numbers. The numbers are arbitrarily chosen and are not contiguous.

`NOJ` gives the serial number of the job episode, always beginning with job number 1. For instance, if an individual in our data set has had three jobs, the data file contains three records for this individual with job number 1, 2, and 3, respectively. Note that only job episodes are included in this data file. If an individual has experienced an interruption between two consecutive jobs, the difference between the ending time of a job and the starting time of the next job may be greater than 1.

`TS` is the starting time of the job episode, in century months. (A century month is the number of months from the beginning of the century; 1 = January 1900.) The date given in this variable records the first month in a new job.

`TF` is the ending time of the job episode, in century months. The date given in this variable records the last month in the job.

`SEX` records the sex of the individual, coded 1 for men and 2 for women.

Box 2 Variables in data file `rrdat.1`

Variable	Column	Description
ID	C1	ID of individual
NOJ	C2	Serial number of the job
TS	C3	Starting time of the job
TF	C4	Ending time of the job
SEX	C5	Sex (1 men, 2 women)
TI	C6	Date of interview
TB	C7	Date of birth
TE	C8	Date of entry into the labor market
TM	C9	Date of marriage (0 if no marriage)
PRES	C10	Prestige score of job i
PRES1	C11	Prestige score of job i + 1
EDU	C12	Highest educational attainment

Box 3 First records of data file `rrdat.1`

ID	NOJ	TS	TF	SEX	TI	TB	TE	TM	PRES	PRES1	EDU
1	1	555	982	1	982	351	555	679	34	-1	17
2	1	593	638	2	982	357	593	762	22	46	10
2	2	639	672	2	982	357	593	762	46	46	10
2	3	673	892	2	982	357	593	762	46	-1	10
3	1	688	699	2	982	473	688	870	41	41	11
3	2	700	729	2	982	473	688	870	41	44	11
3	3	730	741	2	982	473	688	870	44	44	11
3	4	742	816	2	982	473	688	870	44	44	11
3	5	817	828	2	982	473	688	870	44	-1	11

`TI` is the date of the interview, in century months. Using this information, one can decide whether an episode is right censored or not. If the ending time of an episode (`TF`) is less than the interview date, the episode ended with an event, otherwise the episode is right censored.

`TB` records the birth date of the individual, in century months. Therefore, `TS` minus `TB` is the age, in months, at the beginning of a job episode.

`TE` records the date of first entry into the labor market, in century months.

`TM` records whether/when an individual has married. If the value of this variable is positive, it gives the date of marriage (in century months). For still unmarried individuals at the time of the interview, the variable is coded 0.

`PRES` records the prestige score of the current job, that is, the job episode in the current record of the data file.

Box 4 Command file `ed1.cf` (single episode data)

```
nvar(
    dfile = rrdat.1,     # data file

    ID    [3.0] = c1,    # identification number
    SN    [2.0] = c2,    # spell number
    TS    [3.0] = c3,    # starting time
    TF    [3.0] = c4,    # ending time
    SEX   [2.0] = c5,    # sex (1 men, 2 women)
    TI    [3.0] = c6,    # interview date
    TB    [3.0] = c7,    # birth date
    TE    [3.0] = c8,    # entry into labor market
    TMAR  [3.0] = c9,    # marriage date (0 if no marriage)
    PRES  [3.0] = c10,   # prestige of current job
    PRESN [3.0] = c11,   # prestige of next job
    EDU   [2.0] = c12,   # highest educational attainment

    # define additional variables

    COHO1 = ge(TB,348) & le(TB,384),  # birth cohort 1
    COHO2 = ge(TB,468) & le(TB,504),  # birth cohort 2
    COHO3 = ge(TB,588) & le(TB,624),  # birth cohort 3
    W     = SEX[2],                   # women = 1

    DES [1.0] = if eq(TF,TI) then 0 else 1,
    DUR [3.0] = TF - TS + 1,
);
edef(           # define single episode data

    ts = 0,     # starting time
    tf = DUR,   # ending time
    org = 0,    # origin state
    des = DES,  # destination state
);
```

`PRES1` records the prestige score of the consecutive job, if there is a next job, otherwise a missing value (-1) is coded.

`EDU` records the highest educational attainment before entry into the labor market. Lower secondary school qualification (*Hauptschule*) without vocational training is equivalent to 9 years, middle school qualification (*Mittlere Reife*) is equivalent to 10 years, lower secondary school qualification with vocational training is equivalent to 11 years, middle school qualification with vocational training is equivalent to 12 years. *Abitur* is equivalent to 13 years, a professional college qualification is equivalent to 17 years, and a university degree is equivalent to 19 years.

Box 5 Output from `ed1.cf` command (single episode data)

```
> edef(...)
---------------------------------------------------------------------
Creating new single episode data. Max number of transitions: 100.
Definition: org=0, des=DES, ts=0, tf=DUR

                                          Mean
SN  Org Des  Episodes   Weighted   Duration   TS Min    TF Max  Excl
---------------------------------------------------------------------
 1    0   0       142      142.00     128.18      0.00    428.00    -
 1    0   1       458      458.00      49.30      0.00    350.00    -
Sum              600      600.00

Number of episodes: 600
Successfully created new episode data.
```

Box 3 shows the first nine records of data file `rrdat.1`. Note that all dates are coded in *century months*. Thus, 1 means January 1900, 2 means February 1900, 13 means January 1901, and so on. In general:

$$\begin{aligned} \texttt{YEAR} &= \texttt{floor}((\texttt{DATE} - 1) \;/\; 12) + 1900 \\ \texttt{MONTH} &= (\texttt{DATE} - 1) \;\%\; 12 + 1 \end{aligned}$$

where `DATE` is given in century months, and `MONTH` and `YEAR` refer to calendar time. The `floor` operator provides the largest integer less than, or equal to, its argument, and "%" is the *modulus operator*.[1] For instance, the first individual (`ID` = 1) has a single job episode. The starting time is given as century month 555, corresponding to March 1946, and the ending time is 982 = October 1981. Because this is equal to the interview month, the episode is right censored.

Example 1 Command file `ed1.cf` in Box 4 illustrates how to use the example data file `rrdat.1` to define single episode data. The standard output from the `edef` command is shown in Box 5. 458 episodes end in an event, transition from 0 to 1, the mean duration is 128 months; 142 episodes are right censored. Of course, mean duration for censored episodes is no reasonable estimate but might provide useful information about the data. Since we have not used the `cwt` command to define case weights, the number of episodes with and without weights are the same. The column labelled `TSMin` shows the minimum starting time, the

[1] Given two integer numbers, n and m, $n \,\%\, m$ is the remainder after dividing n by m. For instance: $13 \,\%\, 12 = 1$.

Box 6 Command file `ed2.cf` (alternative destination states)

```
nvar(
    dfile = rrdat.1,     # data file

    ID    [3.0] = c1,    # identification number
    SN    [2.0] = c2,    # spell number
    TS    [3.0] = c3,    # starting time
    TF    [3.0] = c4,    # ending time
    SEX   [2.0] = c5,    # sex (1 men, 2 women)
    TI    [3.0] = c6,    # interview date
    TB    [3.0] = c7,    # birth date
    TE    [3.0] = c8,    # entry into labor market
    TMAR  [3.0] = c9,    # marriage date (0 if no marriage)
    PRES  [3.0] = c10,   # prestige of current job
    PRESN [3.0] = c11,   # prestige of next job
    EDU   [2.0] = c12,   # highest educational attainment

    # define additional variables

    COHO1 = ge(TB,348) & le(TB,384),  # birth cohort 1
    COHO2 = ge(TB,468) & le(TB,504),  # birth cohort 2
    COHO3 = ge(TB,588) & le(TB,624),  # birth cohort 3
    W     = SEX[2],                   # women = 1

    # three alternative destination states

    DES [1.0] = if eq(TF,TI) then 0
                else if gt(PRESN/PRES - 1,0.2) then 1
                else if lt(PRESN/PRES - 1,0.0) then 3 else 2,

    DUR [3.0] = TF - TS + 1,
);
edef(            # define single episode data

    ts = 0,      # starting time
    tf = DUR,    # ending time
    org = 0,     # origin state
    des = DES,   # destination state
);
```

column labelled `TSMax` shows the maximum ending time for the corresponding episodes. The final column, labelled `Excl`, is only used when all episodes, for a given origin state, are right censored. There are then no transitions and these episodes will not be used for model estimation.

Command file `ed1.cf` will be the starting point for many examples in the rest of this part of the User's Manual. We simply add further

Box 7 Output from `ed2.cf` command (alternative destination states)

```
> edef(...)
-------------------------------------------------------------------
Creating new single episode data. Max number of transitions: 100.
Definition: org=0, des=DES, ts=0, tf=DUR

                                        Mean
SN  Org Des  Episodes  Weighted   Duration   TS Min    TF Max  Excl
-------------------------------------------------------------------
 1    0   0       142     142.00     128.18      0.00    428.00    -
 1    0   1        84      84.00      45.05      0.00    326.00    -
 1    0   2       219     219.00      45.91      0.00    350.00    -
 1    0   3       155     155.00      56.40      0.00    332.00    -
Sum              600     600.00
```

commands using the episode data defined with the `edef` command.

Example 2 Command file `ed2.cf` in Box 6 illustrate the definition of single episode data with alternative destination states. The command file is almost identical with `ed1.cf`, only the definition of destination states (variable `DES`) has been changed. We now distinguish three different destination states depending on the kind of occupational move.

Part of the standard output from `ed2.cf` is shown in Box 7. 84 episodes end in an upward move, 155 in a downward move, and 219 do not change the prestige level. The number of right censored episodes has, of course, not changed.

This command file file will often be used in later sections to illustrate statistical methods for episodes with alternative destination states.

Example 3 To illustrate the definition of multi-episode data, we use command file `ed3.cf` (not shown). It is almost identical with `ed1.cf`, only the parameters `id=ID` and `sn=SN` have been added to the `edef` command. Part of the standard output is shown in Box 8. The result is typical for multi-episode data. All 201 individual have a first episode, somewhat less have also a second episode, only few have more than three or four episodes. In this example, there are up to 9 episodes, but only a single individual has, in fact, these 9 episodes. Consequently, this way of partitioning episode data is not well suited for statistical analysis. While it would be possible, in principle, to estimate different models for each spell number, this is practically not possible with most multi-episode data sets; there are simply not enough data for higher spell numbers.

Box 8 Part of standard output from `ed3.cf` (multi-episode data)

```
> edef(...)
-------------------------------------------------------------------
Creating new multi-episode data. Max number of transitions: 100.
Definition: id=ID, sn=SN, org=0, des=DES, ts=0, tf=DUR

                                          Mean
SN  Org Des  Episodes  Weighted   Duration   TS Min    TF Max  Excl
-------------------------------------------------------------------
 1    0   0        16     16.00     233.44     0.00    428.00    -
 1    0   1       185    185.00      52.73     0.00    326.00    -
Sum              201    201.00
-------------------------------------------------------------------
 2    0   0        36     36.00     150.17     0.00    407.00    -
 2    0   1       126    126.00      52.30     0.00    350.00    -
Sum              162    162.00
-------------------------------------------------------------------
 3    0   0        38     38.00     121.74     0.00    329.00    -
 3    0   1        69     69.00      50.36     0.00    332.00    -
Sum              107    107.00
-------------------------------------------------------------------
 4    0   0        24     24.00      85.33     0.00    340.00    -
 4    0   1        38     38.00      37.71     0.00    146.00    -
Sum               62     62.00
-------------------------------------------------------------------
 5    0   0         9      9.00     110.22     0.00    328.00    -
 5    0   1        23     23.00      22.61     0.00     70.00    -
Sum               32     32.00
-------------------------------------------------------------------
 6    0   0         8      8.00      45.75     0.00    127.00    -
 6    0   1        12     12.00      43.75     0.00    112.00    -
Sum               20     20.00
-------------------------------------------------------------------
 7    0   0         7      7.00     114.14     0.00    295.00    -
 7    0   1         4      4.00      52.75     0.00     96.00    -
Sum               11     11.00
-------------------------------------------------------------------
 8    0   0         3      3.00      65.00     0.00    172.00    -
 8    0   1         1      1.00      72.00     0.00     72.00    -
Sum                4      4.00
-------------------------------------------------------------------
 9    0   0         1      1.00      34.00     0.00     34.00    *
-------------------------------------------------------------------
Sum              600    600.00
```

3.3.4 Writing Episode Data

Having defined episode data, they can be written into an output file. This is particularly useful for episode splitting. The command is `epdat` with syntax shown in Box 1. All parameters, except the output file name on the right-hand side, are optional. The `epdat` command writes the currently defined episode data into the specified output file; the `ap=1` parameter can be used to append the data to the end of an already existing file. The maximum number of cases can be controlled with the `noc` parameter. As default, the following variables are written:

1. The ID number or, for single episode data, simply the case number, print format: 6.0;
2. The spell number; always 1 for single episode data; print format 3.0;
3. The number of splits; always 1 if without episode splitting; print format 3.0;
4. The current split number; always 1 if without episode splitting; print format 3.0;
5. The origin stae; print format 3.0;
6. The destination state; print format 3.0;
7. The starting time;
8. The ending time. The default print format for the starting and ending time variables is 6.2 but may be changed with the `fmt` parameter).

To add further variables to the output file, one can use the parameter

```
v = varlist,
```

where `varlist` is a list of variables separated by commas. These can be any variables contained in the current data matrix. In particular, one can use type 5 variables defined inside the currently active `edef` command.

As a further option, one can use the `dtda` parameter to request an additional output file containing a description of the output data file. This description follows the TDA syntax for reading data files and can directly be used as a command file.

Box 1 Syntax for `epdat` command

```
epdat (
     noc=...,          maximum number of cases
     fmt=...,          print format for ts and tf, def. 6.2
     v=...,            list of additional variables
     ap=1,             append new data to end of output file
     dtda=...,         TDA description of output file
) = fname;
```

Box 2 Episode data file `ed1.dat`

```
ID SN ORG DES TS TF S1 S2
-------------------------
 1  1   1   2  0 10 11 20
 1  2   2   3 10 12 11 20
 1  3   3   3 12 30 11 20
 2  1   2   1  0 20 15  8
 2  2   1   1 20 35 15  8
```

Example 1 To illustrate the `epdat` command we use the example data `ed1.dat`, shown in Box 2. There are two covariates that will be used for episode splitting. The command file, `ed4.cf`, is shown in Box 3. It first reads the data with an `nvar` command and then defines a single episode data structure. The episodes are split with variables `S1` and `S2`. In addition, the `edef` command defines two time varying dummy variables, `SD1` and `SD2`. They switch from 0 to 1 as soon as the process time reaches the dates given in `S1` and `S2`, respectively.

The resulting output file, `d`, is shown in Box 4. (The header, shown in this box, is not written into the output file.) How the program performs episode splitting should be obvious from this example. For instance, the fourth episode is split two times, first based on `S2 = 8`, then with `S1 = 15`. Also the definition of the time-dependent dummy variables should be easily understandable. For instance, `SD1` goes from 0 to 1 as soon as `S1` becomes greater than, or equal, to the starting time of the current split.

Box 3 Command file `ed4.cf`

```
nvar(

    dfile = ed1.dat,     # data file

    ID    [1.0] = c1,    # identification number
    SN    [1.0] = c2,    # spell number
    ORG   [1.0] = c3,    # origin state
    DES   [1.0] = c4,    # destination state
    TS    [2.0] = c5,    # starting time
    TF    [2.0] = c6,    # ending time
    S1    [2.0] = c7,    # first covariate
    S2    [2.0] = c8,    # second covariate
);

edef(
    ts = TS,    # starting time
    tf = TF,    # ending time
    org = ORG,  # origin state
    des = DES,  # destination state

    SD1   = le(S1,ts), # time-dependent dummy
    SD2   = le(S2,ts), # time-dependent dummy

    split = S1,S2,     # episode splitting
                       # must be the last parameter
);

epdat(                  # write data to d
    v = S1,SD1,S2,SD2,
    dtda = t,
) = d;
```

Box 4 Output file, `d`, created by command file `ed4.cf`

```
ID  SN NSP SPN ORG DES   TS     TF    S1 SD1 S2 SD2
---------------------------------------------------
 1   1   1   1   1   2   0.00  10.00 11  0  20  0
 2   1   2   1   2   2  10.00  11.00 11  0  20  0
 2   1   2   2   2   3  11.00  12.00 11  1  20  0
 3   1   2   1   3   3  12.00  20.00 11  1  20  0
 3   1   2   2   3   3  20.00  30.00 11  1  20  1
 4   1   3   1   2   2   0.00   8.00 15  0   8  0
 4   1   3   2   2   2   8.00  15.00 15  0   8  1
 4   1   3   3   2   1  15.00  20.00 15  1   8  1
 5   1   1   1   1   1  20.00  35.00 15  1   8  1
```

3.3.5 Merging Episode Data

The `ejoin` command can be used to process one or two episode data files. In any case, if episodes are overlapping, they are split in such a way that overlapping parts get appropriate levels. If there are two episode data files, they are merged into one episode data file. The syntax of the command is shown in the following box.

```
ejoin(
    if1=...,        name of first input file
    if2=...,        name of second input file
    max=...,        max block size, def. 1000
    nw=...,         max number of levels, def. 1
    len=...,        max record length, def. 1000
    noc=...,        read maximal noc records, def. all
    fmt0=...,       print format for basic variables
    fmt1=...,       print format for covariates, file 1
    fmt2=...,       print format for covariates, file 2
) = output_file;
```

Input files must be free-format files with at least 6 numerical entries in each record as follows:

1. Case Id used to identify blocks,
2. Number of records (spells) in current block,
3. Record (spell) number in current block,
4. Starting time of spell,
5. Ending time of spell,
6. State (must be a non-negative integer).

Any additional entries are treated as covariates. The number of entries is determined from the first data record in each file.

Note that it is assumed that the input data files are sorted first with respect to the Id variable and, inside each block, with respect to the starting times of the spells.

Each record of the output file will contain the following entries.

1. Case Id,
2. Number of records (spells) in current block,
3. Record (spell) number in current block,
4. Level number (0,1,2,...),
5. Starting time of spell,
6. Ending time of spell,
7. State taken from file 1
8. State taken from file 2 (only if there are two input files).

The print format for these entries can be specified with the `fmt0` parameter. Following these first 7 or 8 entries will be written any covariates that may be present in file 1; the print format for these entries can be controlled with the `fmt2` parameter. Finally follow any covariates from file 2 if there are any; the print format can be controlled with the `fmt2` parameter. If the number of entries in the format parameters is less than required the last one is repeated.

Example 1 To illustrate the `ejoin` command we use the two episode data files shown in Box 1. Both files contain overlapping episodes. The `ejoin` command shown in the same box merges these data files into a single one as shown in Box 2.

Box 1 Two episode data files and a command file (`ej1.cf`)

ej1.dat

ID	NS	SN	TS	TF	S	X1	X2
1	2	1	0	2	1	1.1	11.1
1	2	2	1	5	2	2.2	22.2
2	4	1	3	6	3	3.3	33.3
2	4	2	4	5	4	4.4	44.4
2	4	3	5	8	5	5.5	55.5
2	4	4	9	10	6	6.6	66.6

ej2.dat

ID	NS	SN	TS	TF	S	Y1
1	2	1	1	4	3	5.5
1	2	2	3	6	4	6.6

```
ejoin(
    if1 = ej1.dat,
    if2 = ej2.dat,
    nw = 3,
    fmt0 = 4,
    fmt1 = 8.4,
    fmt2 = 8.4,
) = ej1.d;
```

Box 2 Output file created by `ej1.cf`

ID	NS	SN	LEV	TS	TS	S1	S2	X1	X2	Y1
1	8	1	0	0	1	1	-3	1.1000	11.1000	-3.0000
1	8	2	0	1	2	1	3	1.1000	11.1000	5.5000
1	8	2	1	1	2	2	-3	2.2000	22.2000	-3.0000
1	8	3	0	2	3	2	3	2.2000	22.2000	5.5000
1	8	4	0	3	4	2	3	2.2000	22.2000	5.5000
1	8	4	1	3	4	-3	4	-3.0000	-3.0000	6.6000
1	8	5	0	4	5	2	4	2.2000	22.2000	6.6000
1	8	6	0	5	6	-3	4	-3.0000	-3.0000	6.6000
2	8	1	0	3	4	3	-3	3.3000	33.3000	-3.0000
2	8	2	0	4	5	3	-3	3.3000	33.3000	-3.0000
2	8	2	1	4	5	4	-3	4.4000	44.4000	-3.0000
2	8	3	0	5	6	3	-3	3.3000	33.3000	-3.0000
2	8	3	1	5	6	5	-3	5.5000	55.5000	-3.0000
2	8	4	0	6	8	5	-3	5.5000	55.5000	-3.0000
2	8	5	0	8	9	-3	-3	-3.0000	-3.0000	-3.0000
2	8	6	0	9	10	6	-3	6.6000	66.6000	-3.0000

3.4 Sequence Data

This chapter introduces the concept of sequence data as used in TDA. The subsections are as follows.

3.4.1 Sequence Data Concepts

Sequence data will be defined with respect to a nonnegative discrete time axis $\mathcal{T} = \{0, 1, 2, \ldots\}$ to stress that we are mainly interested in sequences evolving in time. Of course, in some applications as, for instance, in the analysis of DNA sequences, such an intrinsic relationship to time need not be assumed. However, for most applications in the social sciences, we use sequence data as a representation of event-history data which, by definition, evolve in time.

We also assume a finite (discrete) state space, say $\mathcal{Y}$, and use the convention that nonnegative integers represent valid states and negative integers represent non-valid (undefined or missing) states. Note that valid states are restricted to the range 0 – 32000. Given a finite length of the time axis, say T, an individual sequence can then be defined as

$$y_0, y_1, y_2, \ldots, y_T$$

where y_t is a valid or an undefined state at time t.

In most applications, there is not just one single sequence but a sample of sequences. We refer to the sample members as individuals but, of course, any other unit of analysis (which can be identified across time) is possible. Indexing the sample members by $i = 1, \ldots, N$, we then have a sample of sequences

$$y_i = (y_{i0}, y_{i1}, \ldots, y_{iT}) \qquad i = 1, \ldots, N$$

In the following, this will be called a *sequence data structure*. Finally, we have to consider that, for many applications, there can be two or more different sequences for each unit of analysis. We may have data showing how individuals develop in two or more state spaces simultaneously. Or each unit consists of two or more individuals, and we have one or more sequences for each of these individuals. We have then to consider two ore more sequence data structures simultaneously. Using $k = 1, \ldots, K$ to index the sequence data structures, we have

$$y_{ik} = (y_{ik,0}, y_{ik,1}, \ldots, y_{ik,T}) \qquad i = 1, \ldots, N,\ k = 1, \ldots, K$$

y_{ik} denotes the kth sequence for the ith individual. This will be called a K-dimensional sequence data structure. In addition, we use $\mathcal{Y}_k$ to denote the state space for the kth sequence data structure, and $\mathcal{Y} = \mathcal{Y}_1 \cup \cdots \cup \mathcal{Y}_K$.

As said above, we assume that sequences are defined on a nonnegative discrete time axis ($t = 0, 1, 2, \ldots$). This can be a calendar time axis or a process time axis. In both cases, the time axis begins with $t = 0$. The difference is that a calendar time axis does not have a natural origin and, consequently, fixing an origin $t = 0$ is simply a convention for providing time points. On the other hand, the origin of a process time axis has a substantive meaning; $t = 0$ then represents the date of an event providing the entry point of the process.

Example 1 To illustrate, assume we have some monthly measured data on employment status, say 1 = employed, 0 = unemployed, observed for the five-year period 1990 – 1994. We can then define a calendar time axis using the convention 1 = January 1990, 2 = February 1990, and so on. We might have observed two individuals with sequences

```
 1  1  1  0  0  1  1  1  1  ...
-1 -1  1  1  0  0  0  1  0  ...
```

In this example, the two sequences are represented on a calendar time axis. The second individual is only observed beginning in March 1990, so there are two month with missing states. Of course, there may be more missing values throughout and at the end of the sequences.

Alternatively, to represent the data on a process time axis, one has first to decide on an appropriate origin. In this example, this could be the entry into the first (observed) unemployment spell. The sequences for the two individuals would then look as follows:

```
0  0  1  1  1  1  ...
0  0  0  1  0  ...
```

The definition of sequence data for TDA (see **3.4.2**) is independent of the distinction between calendar and process time. The distinction is important, however, when applying statistical procedures.

3.4.2 Defining Sequence Data

Sequences can be defined as a sequence of states or, alternatively, by a sequence of events. TDA supports both methods. The basic command is `seqdef` with syntax shown in Box 1. The command requires a list of variables to be given on the right-hand side. All other parameters are optional.

1. The `sn` parameter can be used to specify the number of the sequence data structure: `sn =` k, where $k = 1, \ldots, 100$. There can be up to 100 simultaneously defined sequence data structures, default is `sn=1`.

2. The `rc` parameter can be used for recoding the states of sequences. This option will be explained below.

3. The `m` parameter can be used to specify the type of input data. There are two possibilities. First, if `m=1` (default), it is assumed that there is a separate variable for each time point. Given that the list of variables on the right-hand side is

`seqdef(...) =` $Y_0, Y_1, \ldots, Y_T$`;`

it is assumed that variable Y_t contains the states for time point t. The sequence data structure will consist of N sequences, one sequence for each case (row) in the currently active data matrix, and each sequence is defined for the time axis $t = 0, 1, \ldots, T$. Alternatively, if `m=2`, it is assumed that the sequence data structure is defined by a sequence of events, specified by an even number of variables in the following way:

`seqdef(...) =` $Y_1, T_1, \ldots, Y_q, T_q$`;`

It is assumed, then, that the ith sequence begins at T_{1i} in state Y_{1i} and ends at T_{qi} in state Y_{qi}. The time variables $T_1, \ldots, T_q$ must contain integer values, and it is required that

$$T_{1i} \geq 0 \quad \text{and} \quad T_{t,i} > T_{t-1,i} \; (t = 2, \ldots, q)$$

If the requirement is not met, TDA will print an error message. For $t < T_{1,i}$ and $t > T_{q,i}$, states are undefined and get the missing value code -1.

Box 1 Syntax for `seqdef` command

```
seqdef (
    sn=...,      number of sequence data structure, def. 1
    m=...,       type of input data, def. 1
    rc=...,      option for recoding states
) = varlist;
```

The time axis. Associated with each sequence data structure is a time axis. For the kth sequence data structure, the time axis begins at $T_k^{\min}$ and ends at $T_k^{\max}$. The definition depends on the sequence data type. If `m=1` and there are state variables $Y_0, Y_1, \ldots, Y_T$, then

$$T_k^{\min} = 0 \qquad \text{and} \qquad T_k^{\max} = T$$

If `m=2` and the variables used to define the sequence data structure are $Y_1, T_1, \ldots, Y_q, T_q$, then

$$T_k^{\min} = \min\{T_{1i} \,|\, i = 1, \ldots, N\}\,,\ T_k^{\max} = \max\{T_{qi} \,|\, i = 1, \ldots, N\}$$

If there are two or more sequence data structures, the range of their common time axis is defined by

$$T^{\min} = \min_k \{T_k^{\min}\} \qquad \text{and} \qquad T^{\max} = \max_k \{T_k^{\min}\}$$

If an individual sequence is undefined for one of the time points, it will get the missing value code -1.

Removing sequence data structures. A sequence data structure defined with the `seqdef` commands remains defined until one of the following conditions occurs:

- A new data structure with the same number is defined by using the `seqdef` command again;
- A variable required to maintain the data structure is deleted with the `clear` command. Note that *all* currently defined sequences will be deleted if at least one required variable is removed with `clear`.
- The sequence data structure is explicitly deleted with the `seqdel` command. This command can be used in two different ways. Using the command

```
seqdel;
```

Box 2 Sequence data file `seq.d1`

```
ID  Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7
--------------------------
 1   7  3  3  1  3  3  7  7
 2   3  3  7  1  1 -1 -1  3
 3  -1  3  3 -1  1  1  1  1
 4   1  1  3  3  3  3 -1 -1
 5   7  3  3  1  3  3  7  7
```

without any arguments deletes all currently defined sequence data structures. Alternatively,

```
seqdel = k;
```

only deletes the sequence data structure k. If this data structure does not exist, the command is ignored. To get information about the currently defined sequence data structures, one can use the command

```
seq;
```

without any arguments. This displays a short table containing some basic information about all currently defined sequence data structures.

Example 1 To illustrate the specification of sequence data we use the data file `seq.d1` shown in Box 2. There are fives sequences, the common sequence length is eight time points. Assuming that the variables are available in TDA's internal data matrix, one can use the command

```
seqdef = Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7;
```

or alternatively

```
seqdef = Y0,,Y7;
```

to define a sequence data structure with default number 1. Complete command files are shown in Box 3.

Part of TDA's standard output when using one of these command files is shown in Box 4. For each sequence data structure, the output shows its number, the type (1 = defined by state variables, 2 = defined by a series of events), the number of (state) variables, the range of the time axis, the number of different states, and finally a list of state numbers.

Box 3 Command file `seq1.cf` using `seq.d1`

```
 first version            a shorter version
---------------          -------------------
nvar(                    nvar(
  dfile = seq.d1,          dfile = seq.d1,
  ID = c1,                 ID = c1,
  Y0 = c2,                 Y{0,7} = c2,
  Y1 = c3,               );
  Y2 = c4,               seqdef = Y0,,Y7;
  Y3 = c5,
  Y4 = c6,
  Y5 = c7,
  Y6 = c8,
  Y7 = c9,
);
seqdef = Y0,,Y7;
```

Box 4 Part of standard output from command file `seq1.cf`

```
Creating a new sequence data structure.
Sequence structure number: 1
Sequence type: 1
Currently defined sequences:

Sequence          State       Time axis       Number
Structure Type  Variables  Minimum  Maximum  of States  States
-------------------------------------------------------------
     1      1         8          0        7         3    1 3 7

Range of common time axis: 0 to 7.
```

Box 5 Sequence data file `seq.d2`

```
     data file                corresponding sequences
---------------------        -------------------------
ID  Y1 T1 Y2 T2 Y3 T3        0  1  2  3  4  5  6  7  8
---------------------        -------------------------
 1   1  0  1  3 -1  4        1  1  1  1 -1 -1 -1 -1 -1
 2   1  0  2  5  2  6        1  1  1  1  1  2  2 -1 -1
 3   2  0  2  6  1  7        2  2  2  2  2  2  2  1 -1
 4   1  0  2  6 -1  7        1  1  1  1  1  1  2 -1 -1
 5   1  4  2  6  3  8       -1 -1 -1 -1  1  1  2  2  3
```

Box 6 Part of standard output using `seq.d2`

```
Creating a new sequence data structure.
Sequence structure number: 1
Sequence type: 2
Currently defined sequences:

Sequence          State       Time axis       Number
Structure Type  Variables  Minimum  Maximum  of States  States
-----------------------------------------------------------------
     1      2         3          0        8         3    1 2 3

Range of common time axis: 0 to 8.
```

Box 7 Structure of a sequence data matrix

```
   Sequence data  Sequence data       Time-independent
    structure 1    structure 2   ...     covariates
  ---------------------------------------------------
1 |             |              |     |                |
  |             |              |     |                |
N |             |              |     |                |
  ---------------------------------------------------
```

Example 2 To illustrate the definition of sequence data based on events, we use the data file `seq.d2` shown in Box 5. The box also shows the corresponding sequences. Again, data are for five cases. There are six variables representing three events. To use this data file for a definition of sequence data, the command should be

```
seqdef(m=2) = Y1,T1,Y2,T2,Y3,T3;
```

(A complete command file is provided as `seq2.cf`.) Part of TDA's standard output is shown in Box 6.

The data matrix. Sequence data are defined by using variables which are available in TDA's internal data matrix. A sequence data structure simply consists of pointers to the internal data matrix; there is no separate copy of the data. In general, for each sequence data structure, the number of sequences equals the number of cases (individuals) in the data matrix. If one wishes to define two ore more sequences for each individual, the whole information for each individual must be contained in a single data matrix row. Box 7 illustrates the data matrix structure.

Box 8 Sequence data file `seq.d3`

```
ID  Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7    S1 T1 S2 T2 S3 T3
-------------------------    ----------------
 1   7  3  3  1  3  3  7  7     1  0  1  3 -1  4
 2   3  3  7  1  1 -1 -1  3     1  0  2  5  2  6
 3  -1  3  3 -1  1  1  1  1     2  0  2  6  1  7
 4   1  1  3  3  3  3 -1 -1     1  0  2  6 -1  7
 5   7  3  3  1  3  3  7  7     1  4  2  6  3  8
```

Box 9 Command file `seq3.cf`

```
nvar(
    dfile = seq.d3,
    ID = c1,
    Y{0,7} = c2,
    S1 = c10,
    T1 = c11,
    S2 = c12,
    T2 = c13,
    S3 = c14,
    T3 = c15,
);
seqdef = Y0,,Y7;    # first sequence data structure
seqdef(             # second sequence data structure
    sn = 2,
    m = 2,
) = S1,T1,S2,T2,S3,T3;
```

Note that some procedures allow to use time-independent covariates in addition to sequence data which, by their nature, are time-varying. The time-independent covariates can simply be specified by referring to some data matrix variables as indicated in Box 7.

Example 3 To illustrate the definition of two parallel sequence data structures, we combine `seq.d1` and `seq.d2` into a single data file, `seq.d3`, as shown in Box 8. Box 9 shows the command file, `seq3.cf`. The `nvar` command creates the data matrix corresponding to the data file, `seq.d3`. The first `seqdef` command uses variables `Y0,...,Y7` to define the first sequence data structure. This command uses the default parameter values `sn=1` and `m=1`. Then follows a second `seqdef` command to define a second sequence data structure (`sn=2`) based on a sequence of events (`m=2`). TDA's standard output from the two `seqdef` commands is shown in Box 10.

Box 10 Part of standard output from `seq3.cf`

```
> seqdef=Y0,,Y7
--------------------------------------
Creating a new sequence data structure.
Sequence structure number: 1
Sequence type: 1
Currently defined sequences:

Sequence          State       Time axis       Number
Structure Type  Variables  Minimum  Maximum  of States  States
-------------------------------------------------------------
     1      1         8          0        7         3    1 3 7

Range of common time axis: 0 to 7.

> seqdef(sn=2,m=2,)=S1,T1,S2,T2,S3,T3
--------------------------------------
Creating a new sequence data structure.
Sequence structure number: 2
Sequence type: 2
Currently defined sequences:

Sequence          State       Time axis       Number
Structure Type  Variables  Minimum  Maximum  of States  States
-------------------------------------------------------------
     1      1         8          0        7         3    1 3 7
     2      2         3          0        8         3    1 2 3

Range of common time axis: 0 to 8.
```

Recoding the state space. By default, values of state variables are used without modification. Nonnegative values represent valid states, and negative values represent undefined states (missing values). The range for valid states is 0 – 32000.

When defining a new sequence data structure with the `seqdef` command, the `rc` parameter can be used to recode the values of the state variables. The syntax is

$$\texttt{rc} = k_1[j_{11}, j_{12}, \ldots], k_2[j_{21}, j_{22}, \ldots], \ldots,$$

Then, if one of the state variables has one of the values $j_{11}, j_{12}, \ldots$, it gets the new value k_1, if it has one of the values $j_{21}, j_{22}, \ldots$, it gets the new value k_2, and so on. Some of the integers in square brackets can be negative to allow a recoding of missing values. It is required, however,

Box 11 Command file `seq4.cf`

```
nvar(
    dfile = seq.d1,
    ID = c1,
    Y{0,7} = c2,
);
seqdef(
    rc = 9[-1],5[7,3],
)   = Y0,,Y7;
seqpd = d;
```

Box 12 Part of output from command file `seq4.cf`

```
Creating a new sequence data structure.
Sequence structure number: 1
Sequence type: 1
Currently defined sequences:

Sequence          State       Time axis       Number
Structure Type Variables  Minimum  Maximum  of States  States
-------------------------------------------------------------
     1      1         8          0        7         3  1 5[3,7] 9[-1]

Range of common time axis: 0 to 7.

Output file d
----------------------------
  ID Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7
   1  5  5  5  1  5  5  5  5
   2  5  5  5  1  1  9  9  5
   3  9  5  5  9  1  1  1  1
   4  1  1  5  5  5  5  9  9
   5  5  5  5  1  5  5  5  5
```

for all values j used inside the square brackets, that

$$-9 \leq j \leq s_{\max}$$

where $s_{\max}$ denotes the highest state number in the original sequence data. It is possible, therefore, only to recode missing values in the range $-1, \ldots, -9$.

Example 4 To illustrate the recode option, we use data file `seq.d1` (Box 2) and command file `seq4.cf`, shown in Box 11. As defined with the `rc` parameter, state -1 is changed into 9, and states 3 and 7 are

changed into 5. Part of the standard output, and the data file `d` created with the `seqpd` command (see section 3.4.3, is shown in Box 12.

Using episode data. Although episode data and sequence data have a quite different data structure, the `seqdef` command allows to directly use single episode data for defining sequence data. Assume there are variables `ORG`, `DES`, `TS`, and `TF`, which can be used to define single episode data with the `edef` command (see **3.3.2**). The same variables can then be used to define sequence data with the command

```
seqdef (m=2) = ORG,TS,DES,TF;
```

Of course, this is no longer possible if the data matrix contains episode splits or multi-episode data. In general, working with sequence data requires that all information about an individual's event history is contained in a single row of the data matrix. However, a command to transform multi-episode data into sequence data will be described in **3.4.4**.

3.4.3 Writing Sequence Data

Given a specification of sequence data, they can be written into an output file. The command is `seqpd` with syntax shown in Box 1. Except for the output file on the right-hand side, all parameters are optional.

1. The `m` parameter can be used to select a structure for the output file. There are four options.

1. If `m=1` (default), the output file will contain a single record for each case in the data matrix, and the sequences will be concatenated horizontally. The first entry in each record is the case number. See the first example in Box 2.

2. If `m=2`, the output file contains a separate record for each sequence data structure. The first entry in each record will contain the case number, the second entry provides the sequence number. See the second example in Box 2.

3. If `m=3`, the output file contains a separate record for each case and time point. The first entry in each record gives the case number, the second entry contains the time point. Then follow the states for all currently defined sequences. This is equivalent to an episode data file where episodes are split on the basis of time points. See the first example in Box 3.

4. If `m=4`, the output file will be an episode data file based on the first currently defined sequence data structure. See the second example in Box 3.

2. The `sel` parameter can be used to select cases. The syntax is

```
sel = expression,
```

Then, only data matrix cases where the result of evaluating `expression` is not equal to zero will be used for the output data file.

3. The `v` parameter with syntax `v=varlist` can be used to write variables into the output file. Values of the variables specified in `varlist` will be written at the end of each output file record. The print format for these variables will be the same as used in their definition. The print

Box 1 Syntax for `seqpd` command

```
seqpd (
    m=...,        structure of output file, def. 1
    sel=...,      expression for case selection
    v=...,        add variables ... to output file
    dtda=...,     request TDA description file
) = fname;
```

Box 2 Illustration of `seqpd` command

```
Command: seqpd (m=1) = d.1;

            first sequence              second sequence
 ID  0  1  2  3  4  5  6  7  8  0  1  2  3  4  5  6  7  8
 -- -------------------------  -------------------------
  1  7  3  3  1  3  3  7  7 -1  1  1  1  1 -1 -1 -1 -1 -1
  2  3  3  7  1  1 -1 -1  3 -1  1  1  1  1  1  2  2 -1 -1
  3 -1  3  3 -1  1  1  1  1 -1  2  2  2  2  2  2  2  1 -1
  4  1  1  3  3  3  3 -1 -1 -1  1  1  1  1  1  1  2 -1 -1
  5  7  3  3  1  3  3  7  7 -1 -1 -1 -1 -1  1  1  2  2  3

Command: seqpd (m=2) = d.2;

 ID SN  0  1  2  3  4  5  6  7  8
 -- -- -------------------------
  1  1  7  3  3  1  3  3  7  7 -1
  1  2  1  1  1  1 -1 -1 -1 -1 -1
  2  1  3  3  7  1  1 -1 -1  3 -1
  2  2  1  1  1  1  1  2  2 -1 -1
  3  1 -1  3  3 -1  1  1  1  1 -1
  3  2  2  2  2  2  2  2  2  1 -1
  4  1  1  1  3  3  3  3 -1 -1 -1
  4  2  1  1  1  1  1  1  2 -1 -1
  5  1  7  3  3  1  3  3  7  7 -1
  5  2 -1 -1 -1 -1  1  1  2  2  3
```

format for the case number is always 6.0; time points are written in a 4.0 format; and the print format for state variables is 2.0.

4. The `dtda` parameter with syntax

```
dtda = name_of_an_output_file,
```

can be used to request an additional output file that will contain a TDA

Box 3 Illustration of `seqpd` command

`seqpd(m=3) = d.3`

ID	T	S1	S2
1	0	7	1
1	1	3	1
1	2	3	1
1	3	1	1
1	4	3	-1
1	5	3	-1
1	6	7	-1
1	7	7	-1
1	8	-1	-1
2	0	3	1
2	1	3	1
2	2	7	1
2	3	1	1
2	4	1	1
2	5	-1	2
2	6	-1	2
2	7	3	-1
2	8	-1	-1
3	0	-1	2
3	1	3	2
3	2	3	2
3	3	-1	2
3	4	1	2
3	5	1	2
3	6	1	2
3	7	1	1
3	8	-1	-1
4	0	1	1
....			
4	5	3	1
4	6	-1	2
4	7	-1	-1
4	8	-1	-1
5	0	7	-1
5	1	3	-1
5	2	3	-1
5	3	1	-1
5	4	3	1
5	5	3	1
5	6	7	2
5	7	7	2
5	8	-1	3

`seqpd(m=4) = d.4`

ID	SN	ORG	DES	TS	TF
1	1	7	3	0	1
1	2	3	1	1	3
1	3	1	3	3	4
1	4	3	7	4	6
1	5	7	-1	6	8
2	1	3	7	0	2
2	2	7	1	2	3
2	3	1	-1	3	5
2	4	-1	3	5	7
2	5	3	-1	7	8
3	1	-1	3	0	1
3	2	3	-1	1	3
3	3	-1	1	3	4
3	4	1	-1	4	8
4	1	1	3	0	2
4	2	3	-1	2	6
4	3	-1	-1	6	8
5	1	7	3	0	1
5	2	3	1	1	3
5	3	1	3	3	4
5	4	3	7	4	6
5	5	7	-1	6	8

description of the data file created with the `seqpd` command. This additional output file can then be used as a command file to read the data file.

Example 1 To illustrate the `seqpd` command, we use the data file `seq.d3` (see section 3.4.2). The command file `seq5.cf` (not shown) is basically identical with `seq3.cf` (see **3.4.2**). We have added four `seqpd` commands, corresponding to the options `m=1,2,3,4`. The output files created by these print commands are shown in boxes 2 and 3.

3.4.4 Deriving Sequences from Episode Data

As mentioned in 3.4.2, single episode data can be used directly to define sequence data. In general, if given an arbitrary multi-episode data file, this is not possible and one needs a procedure to transform episode into sequence data. One such procedure is provided by the `seqpe` command. The syntax is shown in the following box. All parameters, except `v`, are required.

```
seqpe (
    id=...,      ID variable
    org=...,     variable for origin state
    des=...,     variable for destination state
    ts=...,      variable for starting time
    tf=...,      variable for ending time
    tp=...,      definition of time points
    v=...,       add variables ... to output file
) = fname;
```

1. On the right-hand side of the command, one has to give the name of an output file.

2. The parameters `id=`, `org=`, `des=`, `ts=`, and `tf=` must be used to provide variables containing the ID number, origin state, destination state, starting time and ending time, respectively. These variables define the multi-episode data.

3. The `tp` parameter must be used to provide a sequence of time points. The syntax is

$$\text{tp} = t_1, t_2, \ldots, t_n \qquad \text{or} \qquad \text{tp} = t_1\,(d)\,t_2$$

or a mixture of both expressions. Assuming $d > 0$, the second expression is expanded into the sequence $t_1 + id$ $(i = 0, 1, 2, \ldots)$, as long as the result is less than or equal to t_2.

The output file created by the `seqpe` command contains one record for each individual (identified by the `ID` variable). Each record begins

Box 1 Command file `seq6.cf`

```
nvar(
  dfile = d.4,  # data file created with seq5.cf
  noc = 22,
  CASE <5>[6.0] = c1 , # case number
  SN   <2>[2.0] = c2 , # spell number
  ORG  <5>[2.0] = c3 , # origin state
  DES  <5>[2.0] = c4 , # destination state
  TS   <5>[4.0] = c5 , # starting time
  TF   <5>[4.0] = c6 , # ending time
);
seqpe(              # create sequence data
    id = CASE,
    org = ORG,
    des = DES,
    ts  = TS,
    tf  = TF,
    tp  = 0(1)7,
) = d;
```

Box 2 Output file (`d`) create by command file `seq6.cf`

```
ID  0  1  2  3  4  5  6  7
-- ----------------------
 1  7  3  3  1  3  3  7  7
 2  3  3  7  1  1 -1 -1  3
 3 -1  3  3 -1  1  1  1  1
 4  1  1  3  3  3  3 -1 -1
 5  7  3  3  1  3  3  7  7
```

with a case number (always written in 6.0 format). Then follows a sequence of states, the length being equal to the number of time points defined with the `tp` parameter. For each time point t, the algorithm checks whether t falls into one of the episodes, that is, TS $\leq t <$ TF. If successful, the current state of the individual is taken from the variable specified with the `org` parameter, otherwise it is a missing value code (-1). States are written in a 2.0 print format.

4. The `v` parameter can be used to specify a list of variables, printed at the end of each record using their associated print formats. Note that the values of these variables are taken from the first record for each set of episodes belonging to the same individual.

Example 1 To illustrate the `seqpe` command, we use the example

episode data file `d.4`, shown in Box **3.4.3**-3. Remember that this file was created from the first part of sequence data file `seq.d3`. So we can use the `seqpe` command to transform this episode data file back into the original sequence data file (actually `seq.d1`, see **3.4.2**). The command file is `seq6.cf` (Box 1). The `nvar` command reads the variables from the data file `d.4`. The `seqpe` command creates the output file `d` containing the sequences. It is shown in Box 2 and identical with the sequence data file `seq.d1` (see **3.4.2**).

3.4.5 State Indicator Matrices

It might happen that one wants to reorganize a sequence data structure into what has been called a state indicator matrix, for instance, to apply the method of correspondence analysis or some other projection method (see the discussion in Heijden [1987]). The idea is to represent the state of an individual at a specific time point t by a series of q dummy variables, where q is the number of different states. Then, if the individual is in state j at this point in time, the jth dummy variable gets the value one.

To create state indicator matrices, TDA provides the command `seqsi` with syntax shown in Box 1. The command uses the first of the currently defined sequence data structures to create a state indicator matrix and then writes this matrix into the output file specified on the right-hand side of the command. The `tp` parameter must be used to provide a sequence of time points (for a description of the syntax, see **6.5.1**); all other parameters are optional.

1. The `m` parameter can be used to select sequences. If `m=1` (default), the command uses all sequences. If `m=2`, the command uses only sequences having a valid state for all time points defined with the `tp` parameter.

2. The `v` parameter can be used to specify a list of variables. These variables will be added to the records written into the output file.

3. The `dtda` parameter can be used to request a second output file containing an `nvar` command to read the data file created with the `seqsi` command.

The structure of the output file records is as follows:

- The first entry is the case number, written in 6.0 format.
- Then follow $j = 1, \ldots, q$ entries for the first time point; q is the number of different states in the sequence data structure. Entry j is 1 if the sequence is in the jth state at this point in time, otherwise zero. (Note that the states of a sequence are sorted in ascending order and then mapped to internal state numbers: 0,1,2,...; the q entries for each point in time correspond to these internal state numbers.)
- For each additional time point specified with the `tp` parameter there will be again q entries created in an analogous way.

Box 1 Syntax for `seqsi` command

```
seqsi (
    tp=...,      definition of time points
    m=...,       option for case selection, def. 1
    v=...,       add variables ... to output file
    dtda=...,    request TDA description file
) = fname;
```

Box 2 Illustration of `seqsi` command, based on `seq.d1`

```
Command: seqsi(m=1,t=1(1)5,v=ID) = si.d1;
Output file: si.d1

     1 3 7 1 3 7 1 3 7 1 3 7 1 3 7
     ----- ----- ----- ----- -----
Case  t=1   t=2   t=3   t=4   t=5  ID
---- ----- ----- ----- ----- ----- --
   1 0 1 0 0 1 0 1 0 0 0 1 0 0 1 0  1
   2 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0  2
   3 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0  3
   4 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0  4
   5 0 1 0 0 1 0 1 0 0 0 1 0 0 1 0  5

Command: seqsi(m=2,t=1(1)5,v=ID) = si.d2;
Output file: si.d2

     1 3 7 1 3 7 1 3 7 1 3 7 1 3 7
     ----- ----- ----- ----- -----
Case  t=1   t=2   t=3   t=4   t=5  ID
---- ----- ----- ----- ----- ----- --
   1 0 1 0 0 1 0 1 0 0 0 1 0 0 1 0  1
   4 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0  4
   5 0 1 0 0 1 0 1 0 0 0 1 0 0 1 0  5
```

- Finally, any variables specified with the `v` parameter are written at the end of the current record.

Example 1 To illustrate the `seqsi` command, we use the data file `seq.d1` (see 3.4.2) with $q = 3$ states. Box 2 shows the `seqsi` commands (contained in command file `seq7.cf`) and the resulting output files, for a time axis $1, \ldots, 5$ and for both options. With `m=1`, the output file contains five records; with `m=2` there are only three records containing

those sequences which have valid states for all time points. Note the mapping of states to the columns of the output file.

3.4.6 Random Sequences

When experimenting with sequence data, it is sometimes convenient to use random sequences, that is, sequences filled randomly with state numbers. To create such random sequences, TDA provides the command `seqrd` with syntax shown in the following box.

```
seqrd (
    ns=...,      number of states, def. 2
    len=...,     length of sequences, def. 1
    noc=...,     number of sequences, def. 1000
) = fname;
```

The right-hand side must provide the name of an output file. The parameters can be used to specify the number of different states (`ns`), the number of sequences (`noc`), and the length of the sequences (`len`). To create the random states, TDA uses the formula

$$\text{floor}\,(r \cdot \texttt{ns}) + 1$$

where `ns` is the number of different states, and r is a random number which is equally distributed in the $(0, 1)$ interval. For example, the command

```
seqrd (ns=3,noc=100,len=10) = rs.dat;
```

would generate an output file, `rs.dat`, containing 100 sequences each consisting of 10 states randomly selected from $\{1, 2, 3\}$.

3.6 Graphs and Relations

This chapter explains data structures for graphs and relations. It contains the following sections.

3.6.1 Terminology

A simple graph consists of a set of *nodes*, $\mathcal{N}$, and a set of *edges*, $\mathcal{E}$. It will be assumed that the node set is a finite set of positive integers. Two nodes $i, j \in \mathcal{E}$ are said to be connected if there is an edge $(i, j) \in \mathcal{E}$. The edge set may contain edges (i, i), for $i \in \mathcal{N}$; these edges will be called *loops*. (In our terminology, we allow simple graphs to have loops. The main distinction is then between simple graphs and multigraphs, see below. If it is required that a simple graph does not contain loops this will be explicitly mentioned.)

In an *unvalued* graph we only consider whether there is an edge connecting two nodes. Alternatively, each edge may have an associated value which will then be denoted by $v(i, j)$. We shall adopt the convention that only non-negative real numbers are valid values. Consequently, if $v(i, j) \geq 0$, there is an edge connecting nodes i and j having the value $v(i, j)$. On the other hand, if $v(i, j) < 0$, this will mean that there is no edge connecting i and j. (If procedures require valid edges to have positive values this will be explicitly mentioned when describing these procedures.) To represent unvalued and valued graphs we use the notation $(\mathcal{N}, \mathcal{E})$ and $(\mathcal{N}, \mathcal{E}, v)$, respectively.

A graph is called *undirected* if we do not distinguish between the edges (i, j) and (j, i). In a valued graph this will imply that $v(i, j) = v(j, i)$. We then simply speak of a connection between nodes i and j. Alternatively, we may also consider *directed* graphs where we distinguish between (i, j), an edge from i to j, and (j, i), an edge from j to i.

Figure 1 illustrates a simple undirected and unvalued graph consisting of four nodes. There are two edges connecting, respectively, nodes 1 and 2, and 1 and 3. Node 4 is not connected to any other node and is called an *isolated node*.

In a simple graph there is only one kind of edges represented by a single edge set $\mathcal{E}$. More generally, we can consider several kind of edges simultaneously. This will be called a *multigraph*. A multigraph consisting of m simple unvalued graphs will be denoted by

$$(\mathcal{N}, \mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m)$$

Correspondingly, the notation

$$(\mathcal{N}, \mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m, v_1, v_2, \ldots, v_m)$$

Figure 1 Simple undirected and unvalued graph consisting of four nodes and two edges.

will be used to represent a multigraph consisting of m valued simple graphs.

A multigraph allows different interpretations. One interpretation assumes that each edge set represents a different kind of connection between the nodes but that all connections hold simultaneously. Another interpretation views a multigraph as a temporal sequence of simple graphs.

There is a close connection between graphs and relations. Consider a binary relation

$$r : \mathcal{N} \times \mathcal{N} \longrightarrow \mathbf{R}$$

defined on a node set $\mathcal{N}$. We can interpret this relation as representing a graph by using the convention

$$(i, j) \in \mathcal{E} \iff r(i, j) \geq 0$$

This will suffice for an unvalued graph. Of course, we can also think of $r(i, j)$ as representing the value of the edge (i, j). The graph will be undirected if the relation is symmetrical.

3.6.2 Graph Data Structures

Almost all TDA commands that use relational data (graphs) require a previous definition of a *relational data structure* with the `gdd` command. The syntax of this command is shown in Box 1.[1]

1. The command requires a list of variables containing information about node numbers and edges. These variables must be available in a previously defined TDA data matrix (defined with the `nvar` command). What kind of variables are required depends on the option selected with the `opt` parameter. Each of these options will be explained below in turn.

2. Node numbers can be any positive integers. When creating a *gdd data structure*, TDA maps the *external node numbers*, as provided by the user, to *internal node numbers* being a set of contiguous integer numbers, $1, 2, \ldots, N$, where N is the number of nodes of the graph.

3. As will be explained below, the `gdd` command can be used to define multigraphs consisting of several single graphs. Most TDA commands that use relational data provide an optional parameter, `gn` (*graph number*), to select a single graph from the currently defined multigraph. By default, `gn = 1`, that is, the command uses the first graph.

4. The user can specify a *graph type* with the `gt` parameter. The options are as follows:

1. `gt=1` specifies an undirected unvalued graph: an edge from i to j is assumed to exist if the input data contain an edge from i to j *or* an edge from j to i.

2. `gt=2` specifies an undirected valued graph. To make this interpretation unique in cases where the input data contain both, an edge (i, j) and an edge (j, i), there are three options: (a) If `gt(1)=2`, the value of the undirected edge (i, j) is the minimum of $v(i, j)$ and $v(j, i)$. This is the default interpretation and equal to `gt = 2`. (b) If `gt(2)=2`, the value of the undirected edge (i, j) is the maximum

[1] If the `gdd` command is used without any arguments it provides information about the currently defined relational data (if any).

Box 1 Syntax of `gdd` command

```
gdd (
   opt=...,      option, def. 1
                 1 = edge list
                 2 = pointer to adjacency matrix
                 3 = lower triangle
                 4 = upper triangle
                 5 = lower triangle, including diagonal
                 6 = upper triangle, including diagonal
                 7 = full adjacency matrix
   gt=...,       graph type
                 1 = undirected, unvalued
                 2 = undirected, valued
                 3 = directed, unvalued
                 4 = directed, valued
   perm=...,     if perm=1 the gdd data structure is
                 permanently saved, def. perm=0
) = varlist;
```

of $v(i,j)$ and $v(j,i)$. (c) If `gt(3)=2`, the value of the undirected edge (i,j) is the sum of $v(i,j)$ and $v(j,i)$.

3. `gt=3` specifies a directed unvalued graph.

4. `gt=1` specifies a directed valued graph.

If the gdd data structure consists of a multigraph all of its single graphs get an identical graph type as specified with the gt parameter.

5. A relational data structure defined with the `gdd` command remains available for subsequent commands until one of the following conditions occurs:

1. The data structure is explicitly removed with

```
gdd = off;
```

2. The `gdd` command is used again to define a new gdd data structure.

3. At least one of the variables used in the currently defined gdd data structure is no longer available (only if `perm=0`).

4. A new case selection is defined with a `tsel` command (only if `perm=0`).

6. A standard relational data structure defined with the `gdd` command uses the data in TDA's internal data matrix and therefore does not need to store the data again. This saves memory and suffices for most applications. A problem only occurs if one wants to combine a relational data structure with additional variables which characterize its nodes. The best approach would then be to use an additional data file that contains the information (variables) characterizing the nodes of the graph. However, one would then need to set up a new data matrix and this in turn would destroy the previously defined relational data structure.

TDA therefore allows to save a relational data structure permanently, independent of the current data matrix. This can be requested with the `perm=1` parameter. The `gdd` command then creates a copy of the data that are necessary to maintain the relational data structure. Consequently, a relational data structure defined with the `perm=1` option remains defined also if the current data matrix is modified, or substituted by another data matrix. The data structure will only be destroyed if this is explicitly requested with a `gdd=off` command, or if the `gdd` command is used again to define a new relational data structure.

Note, however, that the `perm` parameter can only be used with option 1 (edge list). Otherwise, the command will print a warnings message and the data structure will not be permanently saved.

Option 1: Edge list. For sparse graphs, the most efficient form of data storage is an edge list. If the input data have this form, one can use option 1 with syntax

`gdd =` $I, J, V_1, V_2, \ldots, V_m$`;`

where the right-hand side specifies a list of variable names that must be available in the current data matrix. The variables I and J must contain positive integers and will be used to construct the node set of the graph. The node set will consist of all values found in the variables I or J.

Each of the following variables, $V_1, \ldots, V_m$ specifies the edges of one graph. Altogether, the command allows to specify a multigraph consisting of m simple graphs. There is no specific limit for the number of graphs, m. The interpretation is as follows. If V_k is one of the variables on the right-hand side and V_{ki} denotes its ith value, then

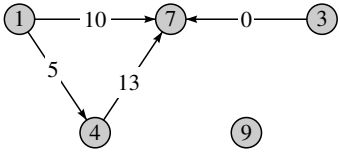


Figure 1 Directed valued graph consisting of five nodes and four edges, see example 1. The plot was created with command file `gd2.cf`.

1. if $V_{ki} \geq 0$ there will an edge connecting nodes I_i and J_i having the (optional) value V_{ki}, and

2. if $V_{ki} < 0$ there will be no edge connecting nodes I_i and J_i.

Note that this conventions allows to specify isolated nodes by giving an edge a negative value. By default, when using option 1, the graph type is 4, i.e., the graph is interpreted as directed and valued.

Example 1 To illustrate, assume we are given the following data:

I	J	V
1	7	10
3	7	0
1	4	5
4	7	13
9	9	-1

The node set will be $\mathcal{N} = \{1, 3, 4, 7, 9\}$. There are four edges, node 9 will be an isolated node. In this example, all four interpretations that can be selected with the `gt` parameter are possible. Figure 1 shows the resulting graph if interpreted as directed and valued (`gt=4`).

Example 2 To illustrate a multigraph we use the data in Box 2. The command

```
gdd = I,J,V1,V2;
```

will create edge lists for both simple graphs. Figure 2 (created with command file `gd3.cf`) provides a graphical illustration.

Box 2 Data file `gd1.dat`

```
 I  J  V1  V2
--------------
 1  5   3  -1
 1  7   5  -1
 7  1   6  15
11 12  13  -1
 9  9  -1  -1
 5  1  -1   4
 8  7  -1   0
12 11  -1  14
 8  8   2  -1
```

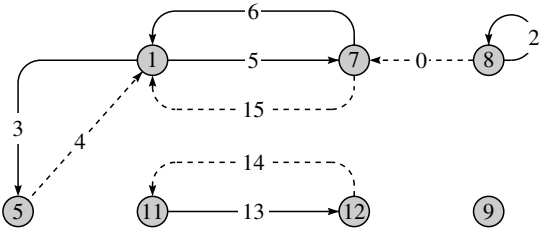


Figure 2 Multigraph consisting of two simple graphs, defined by the edge list shown in Box 2.

Option 2: Pointer to adjacency matrix. If the graph is relatively dense it can be more efficient, and economical, to use pointers to its adjacency matrix. The syntax is again

`gdd =` $I, J, V_1, V_2, \ldots, V_m$`;`

and is interpreted in the same way as already explained for option 1. The difference is only in the internal data interpretation. If `opt = 2`, TDA uses a single $n \times n$ array of pointers to the edge values of the graphs that are contained in the multigraph, n being the maximal number of nodes in the graphs. Of course, there are two drawbacks. If n is large, it might not be possible to store the matrix of pointers with the available memory. Also, some algorithms (in particular, those using some version of a depth-first search) will work less efficient compared with an edge-list data structure. As with option 1, the default graph type interpretation for option 2 is `gt=4` (directed and valued).

Option 3: Lower triangle. Option 3 assumes that the lower triangle of the adjacency matrix is available for input. The syntax is

`gdd =` $V_1, V_2, \ldots, V_m$`;`

It is assumed, then, that each variable V_k, contains the values of the lower triangle of the adjacency matrix for the corresponding graph. To explain the mapping, let $A_k = (a_{k,ij})$ denote the adjacency matrix of the graph corresponding to V_k. Then, for $i > j$, it is assumed that

$$a_{k,ij} = V_{k,l} \quad \text{where} \quad l = \frac{(i-1)(i-2)}{2} + j$$

for $l = 1, \ldots,$ NOC, where NOC is the number of cases in the current data matrix. Note that graphs defined with option 3 are always undirected. By default, `gt=2`, meaning that the graph is interpreted as undirected and valued.

Option 4: Upper triangle. The syntax for option 4 is the same as explained for option 3. The only difference is that the values of V_k are interpreted as entries of the upper triangle of the kth adjacency matrix. Assuming that the graph contains n nodes, and $j > i$, the following mapping is used:

$$a_{k,ij} = V_{k,l} \quad \text{where} \quad l = (i-1)\,n - \frac{i(i-1)}{2} + j - i$$

Again, by default, `gt=2`, that is, the graphs are interpreted as undirected and valued.

Option 5: Lower triangle, including diagonal. The option assumes that V_k contains values for the lower triangle of an adjacency matrix, but includes values of the main diagonal. The syntax is the same as explained for option 3. Assuming that $j \leq i$, the mapping is:

$$a_{k,ij} = V_{k,l} \quad \text{where} \quad l = \frac{i(i-1)}{2} + j$$

By default, `gt=2`, that is, the graphs are interpreted as undirected and valued.

Option 6: Upper triangle, including diagonal. The option assumes that V_k contains values for the upper triangle of an adjacency matrix, but

includes values of the main diagonal. The syntax is the same as explained for option 3. If the number of nodes is n, and $j \geq i$, the mapping is:

$$a_{k,ij} = V_{k,l} \quad \text{where} \quad l = (i-1)n - \frac{(i-1)(i-2)}{2} + j - i + 1$$

By default, `gt=2`, that is, the graphs are interpreted as undirected and valued.

Option 7: Complete adjacency matrix. Finally, if `opt = 7`, it is assumed that V_k contains n^2 values for the complete adjacency matrix, assuming a graph with n nodes. The mapping is

$$a_{k,ij} = V_{k,l} \quad \text{where} \quad l = (i-1)n + j$$

Contrary to options 3 – 6, this option can be used to specify directed graphs. By default, `gt=4`, that is, the graphs are interpreted as directed and valued.

Remark 1 Options 3 – 7 are mainly provided for easy input of proximity data which are most often available as (symmetric) adjacency matrices. No additional data structure will be created if one of these options is selected. The data are directly retrieved from the internal data matrix using the respective mapping described above.

Remark 2 Relational data given by an edge list can be transformed into different versions of an adjacency matrix, and vice versa, by using the `gdp` command described in Section **3.6.4.1**.

3.6.3 Creating Relational Data

This section is intended to describe commands to create relational data having some specified properties. For the moment, there is only a single subsection describing the `gcd` command.

3.6.3.1 Simple Test Data describes the `gcd` command that can be used to create graphs with specified numbers of nodes and edges.

3.6.3.1 Simple Test Data

The `gcd` command can be used to create data files containing unvalued graphs with a specified number of nodes and edges. The syntax is shown in the following box.

```
gcd(
    opt=...,        option, def. 1
                    1 = complete graph, without loops
                    2 = complete graph, including loops
                    3 = randomly select m edges
    n=...,          number of nodes, def. 10
    m=...,          number of edges, def. 1
    nfmt=...,       integer print format, def. 4
) = fname;
```

Except for the name of an output file to be given on the right-hand side, all parameters are optional. The data are written to the output file in form of an edge list. Each record contains values of three variables, I, J, and V. The first two variables contain the node numbers, the third variable, V, has value 1 or -1. If $V_i = 1$, there is an edge from I_i to J_i; otherwise there is no edge.

Option 1. creates a complete graph having n nodes as specified with the `n` parameter. The graph will not contain loops.

Option 2. creates a complete graph having n nodes and includes loops.

Option 3. creates a graph having n nodes and m edges, randomly selected from the set of all edges.

3.6.4 Relational Data Files

This section describes commands intended to support working with relational data. There are two subsections.

3.6.4.1 Writing Relational Data Files describes the `gdp` command that can be used to write the data from a relational data structure into an output file.

3.6.4.2 Creating Adjacency Matrices describes the `mdefg` command that can be used to create a TDA matrix that contains the adjacency matrix of a graph.

3.6.4.1 Writing Relational Data Files

Given a relational data structure, all or part of the data can be written into an output file. The command is `gdp` with syntax shown in the following box.

```
gdp (
    opt=...,        option, def. 1
                    1 = edge list
                    2 = lower triangle of adjacency matrix
                    3 = same as 2 but including the diagonal
                    4 = complete adjacency matrix
                    5 = adjacency matrix written as a square
                    matrix
    gn=...,         graph number (for options 5 and 6), def. 1
    sc=...,         substitute for missing edges, def. -1
    nfmt=...,       integer print format, def. 4
    fmt=...,        print format for values, def. 10.4
    if=...,         input file containing node numbers
) = fname;
```

Except for the name of an output file on the right-hand side, all parameters are optional. One out of six options can be selected to specify the structure of the output file. By default, for options 1 – 4, the `gdp` command writes the edge values for all graphs contained in the currently defined multigraph. Optionally, one can use the `gn` parameter to select a specific graph number. Option 5 writes the adjacency matrix as a square matrix, plus two leading columns that contain, respectively, the internal and external node numbers. By default, this option uses the first graph number if not otherwise specified with the `gn` parameter.

By default, when writing data for a single graph, missing edges are omitted. Of course, they need to be included when writing multigraph data. By default, then, missing edges get the value -1. Alternative values can be specified with the `sc` parameter.

As a further option one can specify the name of an input file with the `if` parameter. The `gdp` command tries to interpret the first numerical

entry in each record as a node number. The command then only writes data for the subgraph induced by the node numbers found in the input file.

One should note that the contents of the output file will depend on the graph type specified in the `gdd` command when defining the relational data structure.

3.6.4.2 Creating Adjacency Matrices

Given a relational data structure, one can use the `mdefg` command to create a TDA matrix containing a graph's adjacency matrix. The syntax is shown in the following box.

```
mdefg (
    gn=...,          graph number, def. 1
    sc=...,          substitute for missing edges, def. -1
) = matrix_name;
```

Except for the name of a matrix to be given on the right-hand side all parameters are optional. By default, the command uses the first graph from the current relational data structure. The `sc` parameter can be used to substitute specific values for missing edges.

3.6.5 Drawing Graphs

There is no easy way to plot arbitrary graphs. TDA therefore does not try to offer a command that automatically plots an arbitrary graph. Instead, there is a command, `plg`, that partially supports plots of graphs based on TDA's environment for creating PostScript plots.

The command requires that the user has defined a PostScript output file with the `psfile` command and set up a coordinate system with the `psetup` command (see 4.1). The `plg` command then allows to place nodes and edges into this coordinate system. For each node one needs a separate `node` parameter, for each edge a separate `edge` parameter. The maximum number of nodes can be specified with the `nmax` parameter, default is 100 nodes. There is no limit for the number of edges.

1. Nodes must be specified as

```
node(...) = x,y,
```

where x and y give the coordinates of the node's center point. All other sub-parameters are optional as explained in Box 1. Here are some additional explanations.

1. Without additional parameters, the `node` parameter plots a circle at the (x, y) position. The diameter can be specified with the `rd` parameter, default is 4 mm.

2. The `gt` parameter can be used to select a different display of nodes. `gt=2` selects a square with diameter defined by `rd`. `gt=4` selects a rectangle. Height and widths can be specified with `rd=h,w`. Default is `h = 4` and `w = 8` (mm). `gt=3` also displays a rectangle but adds two semi-circles.

3. Each node can be given a node number with the `n` parameter. These node numbers are used when an `edge` parameter requires to connect two nodes. Note that there are no default node numbers.

4. Whether a label is written into the node's display depends on the `fs`, `str`, and `n` parameters. If the font size, `fs`, is zero, nothing is displayed. Otherwise, if some string is specified with the `str` parameter, this string is written. (Note that a string must be put in double

Box 1 Syntax of `plg` command

```
plg(
    nmax=...,          maximal number of nodes, def. 100
    node(              place node at (x,y)
          n=...,       external node number, must be integer
          gt=...,      type of display, def. 1
                       1 = circle
                       2 = square
                       3 = rectangle plus semi-circles
                       4 = simple rectangle
          str=...,     optional text if not node number
          gt=...,      type of display, def. 1
          rd=...,      size of display in mm, def. 4 mm
          lt=...,      line type, def. 1
          lw=...,      line width, def. 0.2 mm
          gs=...,      gray scale value, def. 1 (white)
          fs=...,      font size
       ) = x,y,

    edge(              connect nodes i and j
          ic=...,      optional edge value
          fs=...,      font size for edge value
          lt=...,      line type, def. 1
          lw=...,      line width, def. 0.2 mm
          a=...,       size of optional arrow head
          rd=...,      radius for arcs, in mm
          dir=...,     location of loop
       ) = i,j,

);
```

quotation marks to preserve blank characters.) If the `str` parameter is not used, but a node number is specified with the `n` parameter, this node number is written as an integer.

5. The node's display may be grey-scaled with the `gs` parameter. Possible values are in the range from 0 (black) to 1 (white).

2. Edges must be specified as

```
edge(...) = i,j,
```

Box 2 Command file for drawing a graph

```
psfile = gd3.ps;                    PostScript output file
psetup(                             coordinate system
    pxa = 0,9,
    pya = 0,4,
    pxlen = 80,
    pylen = 40,
);
plg(
    node(n= 1,gs=0.8) = 3,3,        plot nodes
    node(n= 7,gs=0.8) = 6,3,
    node(n= 8,gs=0.8) = 8,3,
    node(n= 5,gs=0.8) = 1,1,
    node(n=11,gs=0.8) = 3,1,
    node(n=12,gs=0.8) = 6,1,
    node(n= 9,gs=0.8) = 8,1,

    edge(rd=3,ic=3,a=1.5,1.0) = 1,5,    plot edges
    edge(ic=5,a=1.5,1.0) = 1,7,
    edge(rd=3, ic=6,a=1.5,1.0) = 7,1,
    edge(ic=2,a=1.5,1.0) = 8,8,
    edge(ic=13,a=1.5,1.0) = 11,12,

    edge(lt=5,ic=4,a=1.5,1.0) =  5,1,
    edge(lt=5,rd=-3,ic=15,a=1.5,1.0) = 7,1,
    edge(lt=5,ic=0,a=1.5,1.0) = 8,7,
    edge(lt=5,rd=3, ic=14,a=1.5,1.0) = 12,11,
);
```

where i and j give the numbers of nodes to be connected. Note that only nodes that have a node number specified with the `n` parameter can be connected. (Of course, one can use any of TDA's plot commands to add plot objects.) All other sub-parameters are optional, see Box 1.

1. Without additional parameters, the `edge` parameter connects the two nodes specified by i and j by a straight line. Line type and width can be specified with the `lt` and `lw` parameters, respectively.

2. If $i = j$, the `edge` parameter plots a loop. This will be a circle. Its diameter can be specified with the `rd` parameter, default is 1.5 times the size of the corresponding node. As an additional option, one can use the `dir` parameter to select a place for the loop. There are four possible values, `dir=0,1,2,3`, corresponding to four possible corners.

3. Optionally, two nodes can be connected by a combination of straight

lines and arcs. This can be requested with the `rd` parameter. A positive value specifies the radius of an arc in clockwise direction, a negative value does the same in counter-clockwise direction.

4. An edge may end in an arrow. This can be requested with the parameter

   ```
   a = l,w,
   ```

   where, respectively, `l` is the length and `w` is the width of the arrow head in mm.

5. Finally, one can use the `ic` parameter to specify an edge value (integer or floating point). This value is then written over the edge using the font size defined by the `fs` parameter.

Example 1 To illustrate the `plg` command, Box 2 shows the command file, `gd3.cf`, that was used to create the plot shown in Figure 3.6.2-2.

3.6.6 Trees

An important special case of graphs is called *trees*, defined as connected graphs without cycles. (In this section we always assume undirected graphs.) Alternatively, a tree can be characterized by one of the following equivalent conditions (see Barthélemy and Guénoche [1991, p. 4]):

1. The graph is connected and the number of nodes equals the number of edges plus 1.
2. The graph has no cycles and the number of nodes equals the number of edges plus 1.
3. Any two nodes of the graph are connected by exactly one path.
4. The graph is connected and removing any one of its edges results in an unconnected graph.

As any other graph, a tree can be valued or unvalued. There are two kinds of nodes: nodes having degree one are called *leaves*, and nodes having degree greater than one are called *interior nodes*.

A tree is said to be *rooted*, if one of its nodes is specified as the *root* of the tree. Since any node of a tree can be selected as a root, each tree can be represented by several different rooted trees.

Selecting a root for a tree allows to define a partial order relation on the set of nodes. Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, r)$ denote a rooted tree, $r \in \mathcal{N}$ being the root. For $i, j \in \mathcal{N}$, we can then define: $i \leq j$, if j lies on the path connecting i and r. This defines a partial order relation which is reflexive, antisymmetric, and transitive (see Barthélemy and Guénoche [1991, p. 5]). In addition, we can define $i < j$, if $i \leq j$ and $i \neq j$.

Using this order relation, one can introduce the following terminology. If $i < j$, j is called the *predecessor* of i, and i is called the *successor* of j. This terminology views the tree as originating in its root. Note that each node, except the root, has exactly one predecessor, but may have any number (or zero) of successors.

Example 1 The right part of Box 1 shows data for a valued tree with 13 nodes in the form of an edge list. There are 12 edges connecting nodes `EI` and `EJ` with values given by `EV`. The right part of the box contains a tree data structure that will be explained below. Figure 1 shows a

Box 1 Edge list (left) and tree data structure (right). (Data file `gd4.dat`)

EI	EJ	EV
6	2	1.0
6	4	1.5
7	3	2.0
7	5	2.5
8	1	1.5
8	6	1.0
9	7	3.0
9	8	1.2
1	10	1.6
1	11	2.0
1	12	3.0
7	13	4.0

i	EI	EJ	EV	PR	SU	F
1	6	2	1.5	8	10	6
2	6	4	1.0	6	0	4
3	7	3	2.0	7	0	5
4	7	5	1.5	6	0	0
5	8	1	2.5	7	0	13
6	8	6	1.0	8	2	0
7	9	7	3.0	9	3	8
8	9	8	1.2	9	1	0
9	10	1	0.0	0	7	0
10	11	1	1.6	1	0	11
11	12	1	2.0	1	0	12
12	13	7	3.0	1	0	0
13	0	0	4.0	7	0	0

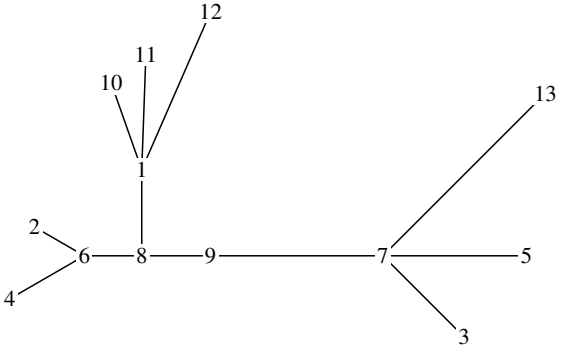


Figure 1 Axial representation of the tree data shown in Box 1.

graphical display of this tree. How to create such plots will be explained below.

Tree Data Structure

Data input for trees is the same as for any other graph, i.e., by an edge list which is then used by TDA to create a gdd data structure. If a procedure expects a tree, it is checked whether the currently defined graph is in fact a tree. If not, the procedure returns with an error message.

For internal representation of rooted trees, TDA follows a proposal of Barthélemy and Guénoche [1991, pp. 5–9] and uses, additionally, a specific data structure that keeps track of the ordering between the nodes, induced by specifying a root node. To explain this data structure, we use the example data in Box 1. The left part of the box shows the input data as an edge list. The right part of the box shows the corresponding tree data structure based on selecting node 9 as a root.

1. The first column (`i`) counts the nodes.

2. The next two columns, labeled `EI` and `EJ` represent the edges of the tree. With n nodes, the tree has $n - 1$ edges and the last row will be empty.

3. The column labeled `EV` contains values of the edges; if the tree is unvalued, all edges will be valued by 1. Note that `EV[i]` corresponds to the edge connecting `i` and `PR[i]`, that is, the predecessor of `i`.

4. The column labeled `PR` contains the predecessor of each node; except the root node where `PR` contains 0.

5. The column labeled `SU` contains zero if the corresponding node does not have a successor, otherwise it contains the number of one arbitrarily selected successor.

6. The column labeled `FR` allows to find all successors of a node. Given node i, the successors of i are given by

`SU[i]`, `FR`[`SU`[`i`]], `FR`[`FR`[`SU`[`i`]]], ...

until the result becomes zero. For instance, node 8 has predecessor 9. One of its successors is given by `SU[8] = 1`, another one by `FR[1] = 6`. Since `FR[6] = 0` there are no more successors of node 8.

The command `ptree`, with syntax shown in Box 2, can be used to print a tree data structure into a file. By default, the command selects an arbitrary node to become the root of the tree. Alternatively, one can specify a root number with the `rt` parameter.

Example 2 To illustrate, we use the `ptree` command to create the data shown in Box 1. Having already build a gdd data structure based on the edge list in the data file `gd4.dat`, the command is

```
ptree (rt = 9) = d;
```

In this example we have selected node 9 to become the root of the tree.

Box 2 Syntax of `ptree` command

```
ptree (
    gn=...,          graph number, def. 1
    rt=...,          root of the tree, def. any
    nfmt=...,        integer print format, def. 4
    fmt=...,         print format for values, def. 10.4
) = fname;
```

Box 3 Syntax of `pltree` command

```
pltree (
    gn=...,          graph number, def. 1
    gt=...,          type of graph (1 or 2), def. 2
    rt=...,          number of root node, def. any
    pl=...,          type of plot (1,2,3), def. 3
    lt=...,          line type, def. 1
    lw=...,          line width in mm, def. 0.2
    fs=...,          font size in mm, def. 2
    nc=...,          label until node ...
    ni=1,            label only for leaves, def. 0
);
```

Graphical Display of Trees

Barthélemy and Guénoche [1991, pp. 23–31] have developed three different methods for a graphical display of trees. These methods are quite useful and have been implemented in TDA. The command is `pltree` with syntax shown in Box 3.

The command requires a PostScript output file, defined with the `psfile` command, and a valid coordinate system, defined with `psetup` (see 4.1). Arbitrary coordinates are possible since the trees are automatically scaled to fit into the given coordinate system.[1] However, the `pltree` command tries to make the physically plotted length of the edges proportional to their values. One should use, therefore, the `pxlen` and `pylen` parameters in the `psetup` command to specify a suitable physical size of the plot.

[1] A simple choice would be to set `pxa=0,1` and `pya=0,1`.

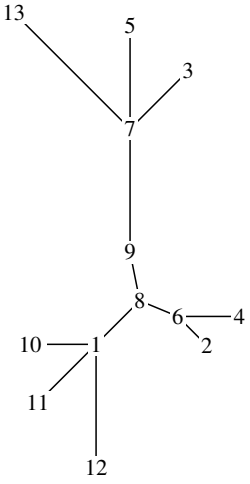


Figure 2 Radial representation of the tree in Box 1.

1. The `gn` parameter can be used to select a graph number, default is `gn=1`.

2. The `gt` parameter can be used to select an interpretation of the graph data. Possible choices are `gt=1` and `gt=2`, default is `gt=2`, that is, the graph is interpreted as undirected and valued.

3. The `lt`, `lw` and `fs` parameters can be used to modify the line type, line width, and font size, respectively.

4. The font size is only used to plot labels (node numbers). If `fs=0`, labels are not plotted. Also, labels are only plotted for nodes $i = 1, \ldots, n$ where n is an integer given with the `nc` parameter. (Note that these are internal node numbers.) By default, `nc=0`, that is, labels are not plotted. In addition, one can use the `ni` parameter. If `ni=1`, labels are only plotted for leaves, not for interior nodes.

5. Three different types of tree drawing can be selected with the `pl` parameter. Radial drawing (`pl=1`), hierarchical drawing (`pl=2`), and axial drawing (`pl=3`). Default is `pl=3`.

Radial drawing. The plot begins at the root of the tree, and then all edges beginning at the root are spread around the root, and then, recursively, the subtrees are plotted, and so on. Figure 1 illustrates this

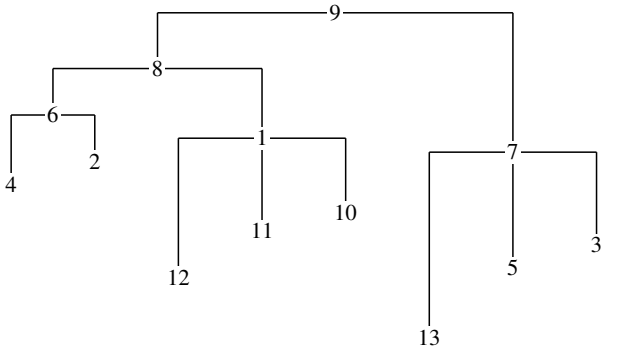


Figure 3 Hierarchical representation of the tree in Box 1.

type of tree plotting with the example data in Box 1.

Hierarchical drawing. The tree is plotted as a hierarchy beginning at the root. Edges are plotted in the y direction of the coordinate system with edge lengths reflecting their values. Figure 2 illustrates this type of tree plotting with the example data in Box 1.

Axial drawing. The root, and as many nodes as possible, are plotted on a line parallel to the x axis. Any remaining nodes and subtrees are then plotted across the available space. Figure 3 illustrates this type of tree plotting with the example data in Box 1. Note that this option requires that the tree is rooted at a node with degree greater than 1. If the procedure is not able to plot the tree for the requested root node it returns with an error message and one should try another root.

Example 3 The example archive contains the command files `gd15.cf`, `gd15a.cf`, and `gd15b.cf`, that have been used to create the plots shown in this section.

4. PostScript Plots

TDA's plot functions are based on the PostScript language, a trademark of Adobe Systems, Inc. This allows for graphics output which is independent of a hardware-specific graphics environment. Creating plots with TDA actually means creating an output file containing a PostScript description of the plot. It will be possible then to send this output file to a PostScript printer to get a hard copy of the plot. Using a PostScript previewer, like GhostScript, it should also be possible to preview plots on a terminal screen.

Moreover, many text processing packages are able to directly include PostScript files. For instance, the documents describing TDA have been written in LaTeX, based on Donald Knuth's TeX. Combined with Tomas Rokicki's DVIPS, a TeX to PostScript driver, one can directly include PostScript files. In fact, almost all figures included in this manual have been created with TDA. To support this possibility, TDA's plot output files follow the conventions for encapsulated PostScript (Version 3.0) as described in Adobe's *PostScript Language Reference Manual* [1990].

4.1 Preparing the Plot Environment explains how to set up the environment for subsequent plot commands.

4.2 Plotting the Coordinate Systems explains how to plot coordinate systems.

4.3 Combining Plot Files shows how to combine separate plot files into a single plot.

4.4 Plot Objects describes commands for a variety of plot objects.

This manual does not provide an introduction to the PostScript language. Actually, it is not necessary to know very much about this graphics language in order to create PostScript plots with TDA. On the other hand, the available commands to control TDA's PostScript output are limited. To make changes, or to access features of PostScript not supported by TDA, one needs to edit the PostScript output files. This is easy but requires, of course, some understanding of the PostScript language.

It is also not intended to give an introduction to computer graphics; this might be found, for instance, in Mortensen [1989], Penna and

Patterson [1986], and Watt [1989]. Useful references discussing the possibilities, and limitations, of using graphical displays in applied statistics are Chambers et al. [1983], and Schnell [1994].

4.1 Preparing the Plot Environment

Creating PostScript plots with TDA actually means to create a file containing a PostScript description of the plot. This is done in three steps.

1. The first step is to create an output file for subsequent plot commands. The command is

```
psfile = name_of_output_file;
```

The file remains opened until the program terminates, the `psfile` command is used again to create a new PostScript output file, or the file is explicitly closed with the command

```
psclose;
```

2. The second step is to set up a coordinate system and, optionally, to modify the physical dimensions of the plot. The command is

```
psetup (parameter);
```

Parameters are shown in Box 1 and will be explained below. The `pxa` and `pya` parameters are required to define a coordinate system. All other parameters are options. Note that the `psetup` command can be used several times with the same output file allowing several plots on the same page.

3. In a third step one can add plot objects by using any of the plot commands that will be described in 4.4.

Parameters for the `psetup` command. Having defined a PostScript output file with the `psfile` command, the `psetup` command must be used to specify a logical coordinate system for subsequent plot commands. The required parameters are `pxa` and `pya`. The syntax is

`pxa =` x_0, x_1`,`

to specify $[x_0, x_1]$ as the range of the x axis, and analogously for the y axis. As default, TDA assumes linear axes. Optionally, one can specify a logarithmic x axis with the parameter

`pxa (log) =` x_0, x_1`,`

Box 1 Syntax for `psetup` command

```
psetup (
     pxa=...,      definition of x axis
     pya=...,      definition of y axis
     pxlen=...,    length of x axis in mm, def. pxlen=120
     pylen=...,    length of y axis in mm, def. pylen=80
     psorg=...,    PostScript coordinates of origin,
                   def psorg=150,460
     psscal=...,   scaling factors, def psscal=1,1
     psrot=...,    rotation, def. psrot=0
);
```

where $x_0 > 0$, and in the same way for the y axis. All other parameters of the `psetup` command are optional.

1. The `pxlen` and `pylen` parameters can be used to modify the physical size of the plot. Default values are `pxlen=120` (mm) and `pylen=80` (mm).

2. The `psorg` parameter can be used to modify the place of the plot in the PostScript coordinate system (see below). Default is `psorg=150,460`.

3. The `psscal` parameter can be used to scale a plot. The syntax is `psscal =` s_x, s_y`,` with s_x and s_y the scaling factors for the x and y axes, respectively. Default is `psscal=1,1`.

4. The `psrot` parameter can be used to rotate the plot in the PostScript coordinate system. Syntax is `psrot =` α, where α is the angle (in degrees) for rotating the plot in counterclockwise direction. Default is `psrot=0`.

Example 1 To give a first example, Box 2 shows the command file `plot1.cf` that was used to create the plot shown in Figure 1. It begins with defining the PostScript output file, `plot1.ps`. Then follows the `psetup` command using the `pxlen` and `pylen` parameters to specify the physical size of the plot, and the `pxa` and `pya` parameters to specify the logical coordinate system. Then follow the `plxa` and `plya` commands to plot the axes, and the `plframe` command to put a frame around the plot. These commands will be explained in 4.2. Finally, there are four plot objects (see 4.4). The `plotf` command plots the `sin` function and the `plotf1` command its first derivative. The two `pltext` commands plot labels.

Box 2 Command file `plot1.cf`

```
psfile = plot1.ps;  output file

psetup(
    pxlen =  90,    # length of x axis in mm
    pylen =  50,    # length of y axis in mm

    pxa   =  0,6,   # user coordinates on x axis
    pya   = -1,1,   # user coordinates on y axis
);

plxa (sc=1,ic=10);  plot x axis
plya (sc=1,ic=0);   plot y axis
plframe;            plot frame

plotf (rx=0(0.1)6) = sin(x);
plotf1(rx=0(0.1)6) = sin(x);

pltext(xy=2.7,0.6) = sine ;
pltext(xy=4.6,0.6) = cosine ;
```

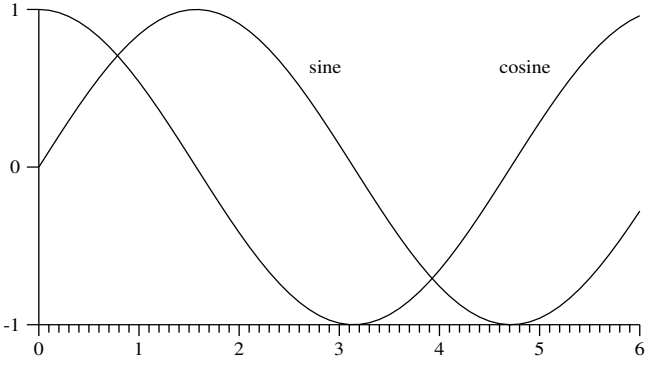


Figure 1 Example of a PostScript plot, created with command file `plot1.cf` shown in Box 2.

PostScript Coordinate System. As mentioned, creating PostScript plots requires the specification of a coordinate system. There are two kinds of coordinates: logical and physical coordinates. All plot objects are defined with respect to a system of logical coordinates, specified with the `pxa` and `pya` parameters in the `psetup` command. A physical coordinate system is used to map the logical coordinate system onto a

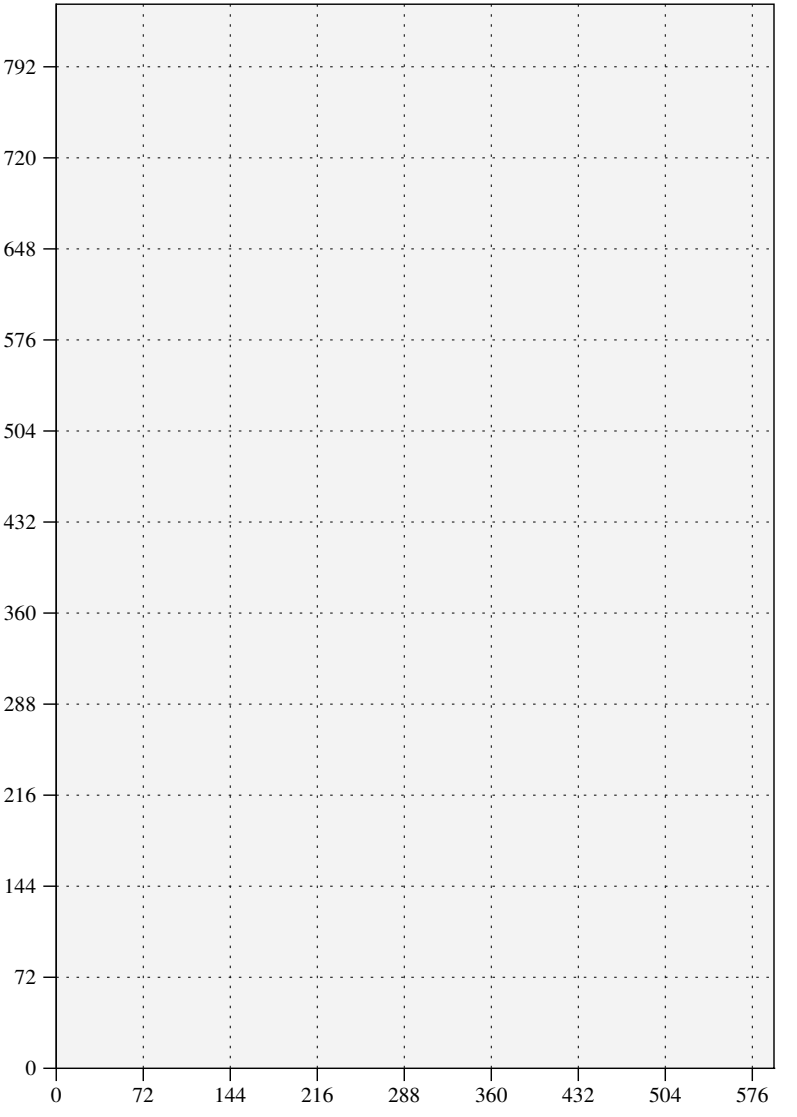


Figure 2 PostScript coordinate system corresponding to a DIN A4 page, created with command file `plot2.cf` (not shown).

two-dimensional space (a sheet of paper or the screen of a terminal).

The physical coordinate system of the PostScript language is defined in *points* (1 inch = 72 points). TDA uses mm (millimeter) and calculates with 1 mm = 2.835 points. On a standard DIN-A4 page the origin is at the lower left corner. The x dimension has 595 points, the y dimension has 842 points. Figure 2 illustrates this coordinate system; the plot was created with command file `plot2.cf` which is contained in the TDA example archive.

PostScript Bounding Box. TDA creates encapsulated PostScript files as described in Adobe's PostScript Language Reference Manual [1990]. This requires the whole plot to be encapsulated into a *bounding box* to be defined in PostScript coordinates. This then allows the PostScript output file to become part of other PostScript documents.

TDA tries to create a bounding box being sufficiently large to include all plot objects. There may occur situations, however, when this cannot be done. For instance, if the whole plot is rotated with the `psrot` command, the default bounding box is not adjusted in a proper way. Another situation where the bounding box is not sufficiently large can occur when the `pltext` command has been used to place text outside the coordinate system. However, it is easy to adjust the bounding box manually. Two different methods can be used. First, one can use the `plrec` command to place rectangles (having zero size) at any desired position which then becomes part of the bounding box. Alternatively, one can directly change the PostScript coordinates of the bounding box in the PostScript output file.

4.2 Plotting Coordinate Systems

The `pxa` and `pya` parameters that are part of the `psetup` command (see section 4.1) only define a logical coordinate system; they do not create any graphical output. To plot axes of a coordinate system, one can use the `plxa` and `plya` commands for the x and the y axis, respectively. Both commands have the same syntax as shown in the following box.

```
plxa(
     sc=...,        distance of main tick marks
     ic=...,        number of sub-intervals
     lt=...,        line type, def. 1 (solid line)
     lw=...,        line width, def. 0.2 (mm)
     fs=...,        font size for labels, def. 2 (mm)
     fmt=...,       print format for labels, def. 0.0
     tl=...,        length of tick marks, def. 1.8 (mm)
     dir=...,       direction of axis, def. 0
) = xa,ya,xb,yb;
```

1. All parameters are optional. Without any parameters, the command plots simply the axis, without labels, beginning at the lower left corner of the coordinate system. The `lt` and `lw` parameters can be used to control the line type and line width, respectively. Default is `lt=1` (solid line) and `lw=0.2` (mm).

2. To specify a different place for the axes, one can use the right-hand side parameters. The axis is then plotted from `(xa,ya)` to `(xb,yb)` (logical coordinates).

3. As default, the x and y axes are not labeled. To get numerical labels, one has to define tick marks. This can be achieved with the optional `sc` parameter. Given `sc=d`, this creates a series of main tick marks placed at

$$\texttt{xa} + i\,\texttt{d} \quad i = 0, 1, 2, \ldots$$

until `xb` is reached. These main tick marks also get numerical labels. The font size can be controlled with the `fs` parameter. The size is in mm;

Box 1 Command file (`plot3.cf`)

```
psfile = plot3.ps;  output file
psetup(
    pxlen =  90,    # length of x axis in mm
    pylen =  50,    # length of y axis in mm

    pxa   = -2,2,   # user coordinates on x axis
    pya   =  0,1,   # user coordinates on y axis
);
plxa (sc=1,ic=2);     plot x axis
plya (sc=0.5,ic=0);   plot y axis
plframe;               plot frame

plxgrid = 0.5;
plygrid = -1,0,1;

plabel  = "Linear Coordinate System";
pxlabel = "X axis";
pylabel(sc=3) = "Y axis";   shift 3 mm to right
```

default font size is `fs=2` (mm). To suppress labels, one can use `fs=0`. The print format for the labels can be controlled with the `fmt` parameter; default is a free format. The `tl` parameter controls the length of the main tick marks (in mm); default is `tl=1.8` (mm).

4. In addition, one can request small tick marks with the `ic` parameter. Given the parameter `ic=n`, each interval between two main tick marks is subdivided into `n` subintervals.

5. Finally, one can use the `dir` parameter to control the direction of the axes. If `dir=0` (default), labels for the x axis are placed below, and labels for the y axis are placed on the left-hand side of the axis. When using the `dir=1` parameter the tick marks and labels are plotted in the opposite direction. Note that the `plxa` and `plya` commands can be used more than once, see Example 2.

Additional Options. There are a few additional commands to control the graphical display of the coordinate system. The Command

```
plframe (lt=,lw=,gs=);
```

can be used to plot a frame around the coordinate system. The parameters are optional. The `lt` and `lw` parameters can be used to control the line type and line width, respectively. Default is always `lt=1` (solid line)

Box 2 Command file (`plot4.cf`)

```
psfile = plot4.ps;  output file
psetup(
    pxlen =  90,
    pylen =  50,

    pxa(log=1) = 1,1000,
    pya(log=1) = 2,500,
);
plxa (sc=10,ic=10);
plya (sc=5,ic=0);
plframe;

plxgrid = 10,50,250;
plygrid = 10,100,1000;

plabel  = "Logarithmic Coordinate System";
pxlabel = "X axis";
pylabel(sc=3) = "Y axis";   shift 3 mm to right
```

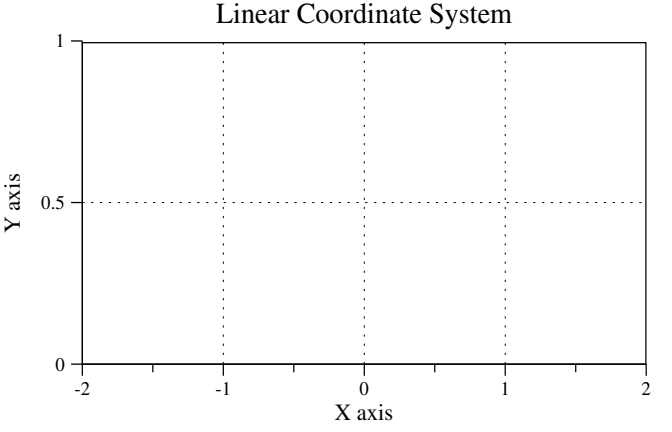


Figure 1 Plot of `plot3.ps`, created by command file `plot3.cf` shown in Box 1.

and `lw=0.2` (mm). The parameter `gs=x` can be used to fill the rectangle with a grey tone. `x` must be a real value in the range from 0 (black) to 1 (white).

The `plxgrid` and `plygrid` commands can be used to put a system

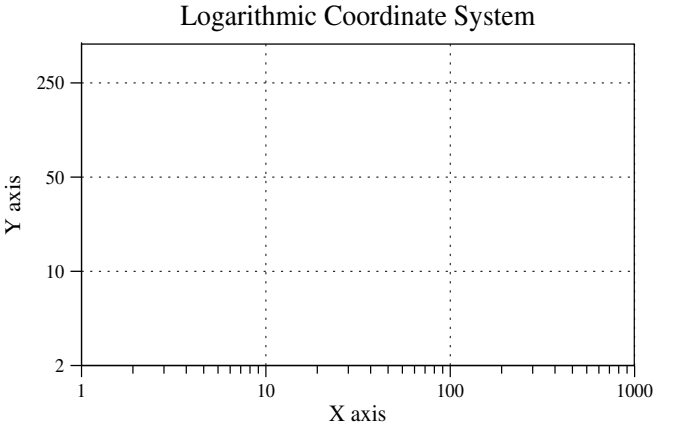


Figure 2 Plot of `plot4.ps`, created by command file `plot4.cf` shown in Box 2.

Box 3 Command file `plot5.cf`

```
psfile = plot5.ps;  output file
psetup(
    pxlen =  90,
    pylen =  50,
    pxa   = -2,2,
    pya   =  0,1,
);
plxa (sc=1,ic=2);     plot x axis
plya (sc=0.5,ic=0);   plot y axis

plxa(sc=1,ic=5,dir=1) = -1,1,1,1;   plot another x axis
plya(sc=0.5,ic=5,dir=1) = 2,0,2,1;  plot another y axis
```

of grid lines onto the coordinate system. The syntax for the `plxgrid` command is

```
plxgrid (lt=,lw=) = y1,y2,... ;
```

This places horizontal grid lines (parallel to the X axes) at the logical Y coordinates `y1,y2,...` The default line type is `lt=3` (dotted), and the default line width is `lw=0.05` (mm). The `plygrid` command has an analogous syntax.

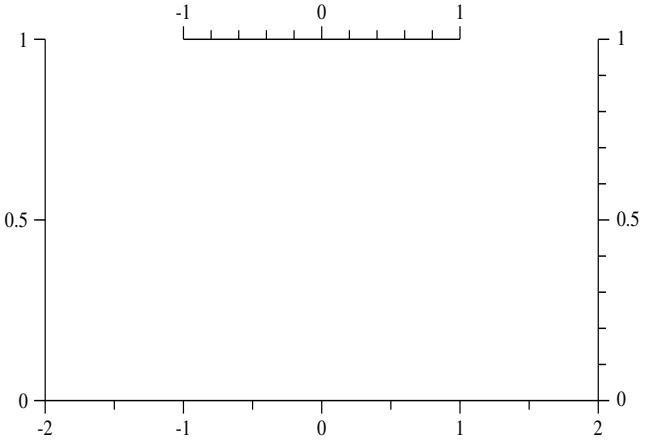


Figure 3 Plot of `plot5.ps`, created by command file `plot5.cf` shown in Box 3.

Three further commands can be used to plot labels. The command

```
plabel (sc=,fs=) = string ;
```

places `string` at the top of the coordinate system to provide a label for the whole plot. The default font size is `fs=3` (mm). The parameter `sc` (default `sc=0` mm) can be used to change the distance between the top of the coordinate system and the label. `string` should be enclosed in (single or double) quotation marks to maintain blank characters, and should only contain characters from a single font (see **4.4.3**). If one needs labels containing characters from different fonts, one should use the `pltext` command.

Analogously, one can use the commands `pxlabel` and `pylabel` to define labels for the x and y axes, respectively. The syntax is identical with the `plabel` command; default font size is `fs=2.4` (mm).

Example 1 Command file `plot3.cf`, shown in Box 1, illustrates how to plot a linear coordinate system. The resulting PostScript plot, written into the output file `plot3.ps`, is shown in Figure 1. Command file `plot4.cf` (Box 2) illustrates a logarithmic coordinate system; the resulting plot is shown in Figure 2.

Example 2 The `plxa` and `plya` commands can be used several times to place different axes into the same plot. An example, `plot5.cf`, is shown in Box 3. The resulting plot is Figure 3.

Box 4 Command file `plot6.cf`

```
psfile = plot6.ps;  output file
psetup(
    psorg = 100,600,    # set physical origin
    pxlen =  50,
    pylen =  40,
    pxa = 0,6,
    pya = -1,1,
);
plxa(sc=1);
plya(sc=1);

plotf(rx=0(0.1)6) = sin(x1);   add a plot object
pltext(xy=4,0.2) = "Plot 1";   give a label

psetup(                 # define a new coordinate system
    psorg = 270,600,    # set physical origin
    pxlen =  50,
    pylen =  30,
    pxa = 0,6,
    pya = -1,1,
);
plxa(sc=1);
plya(sc=1);

plotf(rx=0(0.1)6) = sin(x1);   add a plot object
pltext(xy=4,0.2) = "Plot 2";   give a label
```

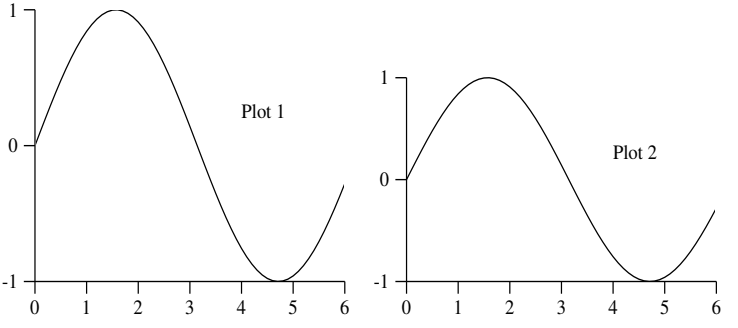


Figure 4 Example of multiple coordinate systems in a single PostScript file. The plot was created with command file `plot6.cf` shown in Box 4.

Multiple Coordinate Systems. The `psetup` command can be used several times with the same PostScript output file to allow multiple coordinate systems in a single plot. For properly placing the coordinate systems, one should use the `psorg` parameter; for an orientation see Figure 4.1-2.

Example 3 Command file `plot6.cf`, shown in Box 4, provides an example of using two coordinate systems for a single PostScript output file. The first `psetup` command creates a coordinate system having its lower left corner at PostScript coordinates (100,600). All subsequent plot commands use this coordinate system, until a new one is created with the second `psetup` command. The remaining plot commands use this second coordinate system. Figure 4 shows the resulting plot. The relative size and position of the two parts is determined by their physical size and origins. Note that the origins, that is, the lower left corner of the physical coordinate systems, must be specified with reference to the PostScript coordinate system, see section 4.1.

4.3 Combining Plot Files

The `dplot` command can be used to combine a set of PostScript files into a single file. This command is independent of all other plot commands and in particular does not require the `psfile` and `psetup` commands. The syntax is shown in the following box.

```
dplot (
     fn=...,        list of input files
     pxlen=...,     x-size of output plot in mm, def. 120
     pylen=...,     y-size of output plot in mm, def. 80
     psorg=...,     origin of output plot, def. psorg=10,10
) = fname;
```

The right-hand side must provide the name of an output file, and the `fn` parameter must be used to specify at least one set of PostScript files previously created with TDA. These files are placed in one row of plots in the output file. If used more than once (up to 50), each set of files is placed into a separate row of plots.

The other parameters are optional. The `pxlen` and `pylen` parameters can be used to modify the physical size of the resulting output plot. The `psorg` parameter can be used to modify the origin of the output plot; syntax is `psorg=x,y` where `x` and `y` are PostScript coordinates.

Box 1 Command file `plot7.cf`

```
dplot(

    pxlen = 100,          #  length of x axis in mm
    pylen =  60,          #  length of y axis in mm
    fn = plot3.ps, plot3.ps,
    fn = plot4.ps, plot4.ps,

) = plot7.ps;           # name of output file
```

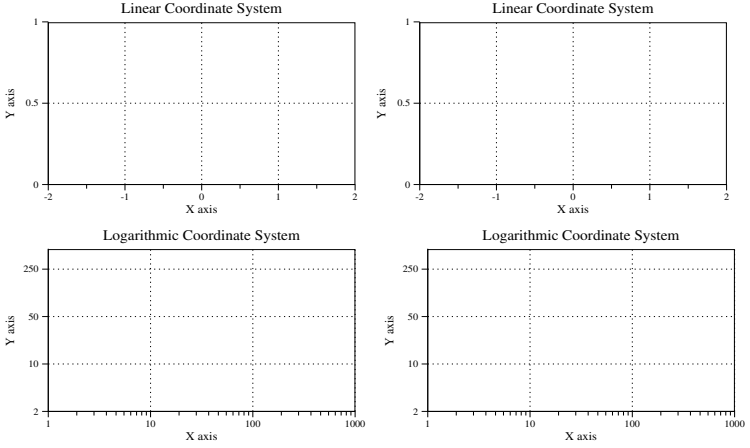


Figure 1 PostScript plot `plot7.ps`, created with command file `plot7.cf` shown in Box 1.

Example 1 To illustrate the `dplot` command, Box 1 shows the command file `plot7.cf`. This command files combines four plots and writes the result into the output file `plot7.ps`. The resulting plot is shown in Figure 1.

While this example works fine, problems can occur when the relation between the physical origin and the bounding box is not the same in all input files. There is, in general, no safe automatic way to arrange the individual plots such that the individual coordinate axes are always in line. If such problems occur, one has to edit the final output file and adjust the `xorg` and `yorg` commands.

4.4 Plot Objects

This chapter describes commands for a variety of standard plot objects. Further plot commands may also be found in other parts of the manual.

4.4.1 Polygons and Scatterplots

There are two commands to plot an arbitrary sequence of data points, `plot` and `plotp`. The only difference is in the way the data points are supplied. With the former one, the data points are taken from the program's internal data matrix, with the latter one, the data points can be explicitly supplied as part of the command.

One important application of these commands is to create scatterplots. However, the same command can be used to plot a line, a rectangle, or an arbitrary function, $y = f(x)$, given by a sequence of points, (x_i, y_i). The syntax of the first command, `plot`, is shown in Box 1. The right-hand side of the command must contain exactly two variable names, to be used for the x and y coordinates of the data points. All other parameters are optional.

1. By default, the command uses all cases (rows) of the currently defined data matrix. To select a subset of cases one can use the `sel` parameter with syntax

```
sel = expression;
```

Given any valid `expression` which may contain names of variables, constants and operators, the command uses only those cases of the data matrix where the evaluation of `expression` results in a value not equal to zero. Note that there must be at least two data points.

2. The parameters `lt` and `lw` can be used to change the line type and line width, respectively. Defaults are `lt=1` (solid line) and `lw=0.2` (mm); see the description in 4.4.2.

3. The `s` parameter can be used to request marker symbols to be plotted at the location of the data points. The syntax is `s=n` where `n` is the number of one of the available marker symbols, see 4.4.2. The size of the markers symbols can be controlled with the `fs` parameter; default is `fs=2` (mm).

4. By default, the data points are connected by straight lines. To suppress these lines, one can use `lt=0` or `lw=0`. However, the line width defined by the `lw` parameter is also used for the marker symbols. There-

Box 1 Syntax for `plot` command

```
plot (
     lt=...,        line type, def. 1 (solid line)
     lw=...,        line width, def. 0.2 (mm)
     s=...,         marker symbol
     fs=...,        size of marker symbol, def. 2 (mm)
     gs=...,        grey scale value, def. 1 (white)
     r=...,         creates step function
     dir=...,       type of step function
     ns=...,        if ns=1 plot only horizontal lines, def. 0
     a=...,         creates an arrow
     nc=...,        no clipping option, def. 0
     sel=...,       selection of data points
) = VX,VY;
```

Box 2 Syntax for `plotp` command

```
plotp (
     lt=...,        line type, def. 1 (solid line)
     lw=...,        line width, def. 0.2 (mm)
     s=...,         marker symbol
     fs=...,        size of marker symbol, def. 2 (mm)
     gs=...,        grey scale value, def. 1 (white)
     r=...,         creates step function
     dir=...,       type of step function
     ns=...,        if ns=1 plot only horizontal lines, def. 0
     a=...,         creates an arrow
     nc=...,        no clipping option, def. 0
) = x1,y1,...,xn,yn;
```

fore, to get only the marker symbols without connecting lines, one should use `lt=0`.

5. The `r`, `dir` and `ns` parameters can be used to request step functions and histograms as will be explained in 4.4.5.

6. The `a` parameter can be used to request that the polygon ends in an arrow, see 4.4.7.

7. The `gs` parameter can be used to fill the area between the x axis and

Box 3 Command file `plot8.cf`

```
nvar(
    noc = 20,           # create data
    RD  = rd(-2,2),
    RDS = sort(RD),
    ND  = nd(RDS),
);
psfile = plot8.ps;  output file
psetup(
    pxlen =  90,
    pylen =  50,
    pxa   = -2,2,
    pya   =  0,1,
);
plxa (sc=0.5,fmt=4.1);  plot x axis
plya (sc=1,ic=10);      plot y axis
plframe;                plot frame

plot (s=4,fs=1.5) = RDS,ND;   polygon with marker symbols
```

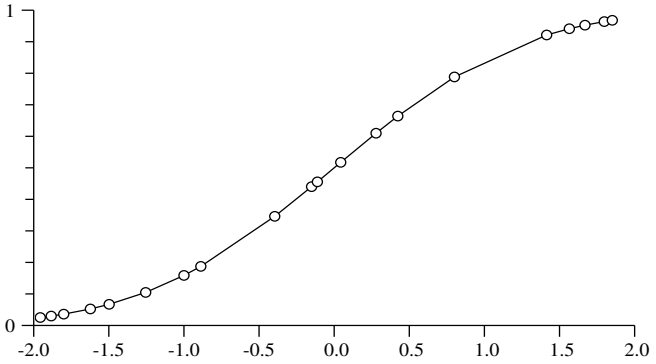


Figure 1 PostScript plot `plot8.ps`, created with command file `plot8.cf` shown in Box 3.

the polygon with a grey tone. The syntax is `gs=x` with `x` a real value between 0 (black) and 1 (white).

8. By default, the plot is restricted to the area defined by the currently active coordinate system. Any part of the plot object that falls outside this area is clipped. To allow for plotting outside this area one can use the command `nc=1`. The program then also tries to adjust the bounding

Box 4 Syntax for `plotm` command

```
plotm(
     lt=...,          line type, def. 1 (solid line)
     lw=...,          line width, def. 0.2 (mm)
     s=...,           marker symbol
     fs=...,          size of marker symbol, def. 2 (mm)
     gs=...,          grey scale value, def. 1 (white)
     r=...,           creates step function
     dir=...,         type of step function
     a=...,           creates an arrow
     nc=...,          no clipping option, def. 0
     sel=...,         selection of data points
) = X1,Y1,X2,Y2,...;
```

box of the plot properly.

The syntax of the second command, `plotp`, is shown in Box 2. The right-hand side is a sequence of `n` data points to be given in user coordinates. There must be at least two data points. All other parameters are optional and have the same meaning as explained above.

Example 1 To illustrate, command file `plot8.cf`, shown in Box 3, generates 20 equally distributed random numbers in the range $(-2, +2)$. These random numbers are sorted and taken as arguments of the standard normal distribution function. As shown by the `plot` command, the data points with coordinates given by variables `RDS` and `ND` are plotted using a marker symbol of type 4 and size 1.5 mm; in addition, the points are connected by a sequence of straight line segments. The resulting plot is shown in Figure 1.

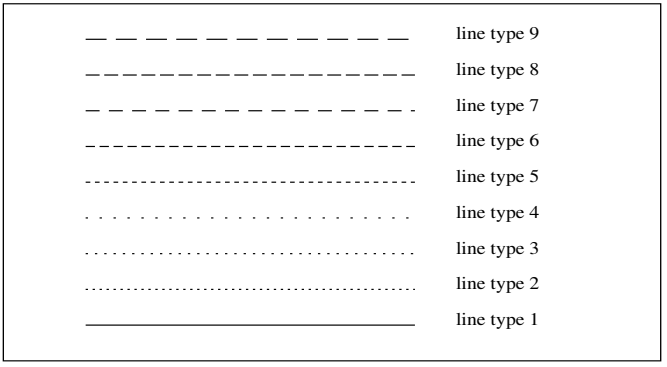
Separate Polygons. The `plotm` command can be used to plot a polygon (or scatter plot) separately for each row in the data matrix. The syntax is shown in Box 4. The right-hand side must provide at least two pairs of variables: `X1,Y1,X2,Y2,...`. The command uses the values of these variables as coordinates, separately for each row in the data matrix, meaning that the ith plot is based on

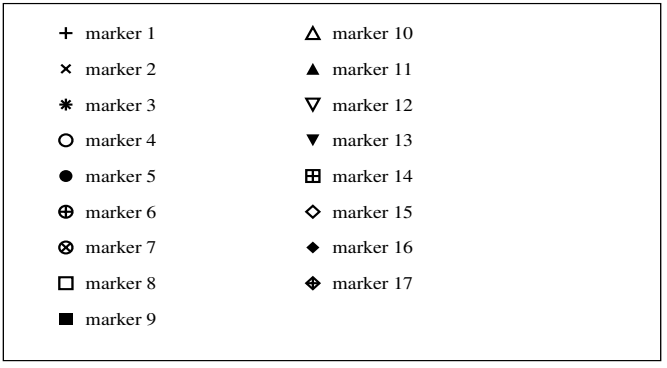
```
(X1(i),Y1(i)), (X2(i),Y2(i)), ...
```

where i refers to the ith row in the data matrix. All parameters are optional and have the same meaning as explained for the `plot` command.

4.4.2 Line Types and Marker Symbols

Many plot commands have an `lt` parameter to select a line type and an `s` parameter to select a marker symbol. The available options are shown in the following boxes.





Note that there is an additional line type: zero, meaning that no line is drawn to connect two data points. This is a useful option if, for instance, data points shall be plotted by marker symbols without connecting lines.

A zero line type can also be used to create grey-scaled plot objects without visible bounding line segments.

4.4.3 Plotting Text

To plot an arbitrary text, one can use the `pltext` command with syntax shown in the following box.

```
pltext (
     xy=...,        coordinates
     fs=...,        font size, def. 2 (mm)
     sc=...,        sc=1 centers the string, def. 0
     r=...,         rotation, def. 0
     s=...,         number of a marker symbol
) = string;
```

The string given on the right-hand side is plotted beginning at the coordinates `(x,y)` to be given with the `xy=x,y` parameter. Note that strings should be enclosed in single or double quotation marks to preserve blank characters.

All other parameters are optional. The `fs` parameter can be used to change the font size; the default is 2 mm. The parameter `sc=1` can be used to center the string around the point `(x,y)`. The `r` parameter can be used to rotate the string; `r=d` rotates the string by `d` degrees in counterclockwise direction; default is `r=0`. Finally, one can use the `s` parameter to provide the number of a marker symbol. This symbol is then plotted in front of the string. Note that this option implies that the string cannot be centered and/or rotated.

Fonts. The PostScript language provides a variety of different fonts. In TDA, two standard fonts are used for strings defined with the `pltext` and/or `plabel` commands: Times-Roman and Symbol. It is assumed that both fonts are available on almost all PostScript devices. Figures 1 and 2 show the available characters and symbols for these two fonts. (We are grateful to Wolfgang Voges (Bremen) who has provided a convenient recoding of these fonts containing a lot of useful characters and symbols and has shown us how to include these fonts into TDA's PostScript features.) To reference these characters and symbols, one should use

```
\nnn   and   @nnn
```

for Times-Roman and Symbol fonts, respectively. In both cases, `nnn` is the character's octal code as shown in the top left corner of each character's box.

Example. Command file `plots.cf`, shown in Box 1, illustrates possible usage of special characters and symbols. The resulting plot is shown in Figure 3.

Figure 1 Times-Roman fonts in TDA encoding.

Figure 2 Symbols in TDA encoding.

Box 1 Command file `plots.cf`

```
psfile = plots.ps;  output file
psetup(
  pxlen = 120,
  pylen = 60,
  pxa = 0,10,
  pya = 0,5 ,
);
plxa (sc=1);
plya (sc=1,ic=2);
plframe;

pltext(xy=1,4) = "@141 = 2.5 \245 @142";
pltext(xy=4,4) = "pltext (xy=1,4) =
                              \324\100141 = 2.5 \\245 \100142\324";
pltext(xy=1,2) = "@123 = @173 @161 @174 @161 @316 @121 @175";
pltext(xy=2.2,0.9) = "pltext (xy=1,2) = \324\100123 =
      \100173 \100161 \100174 \100161 \100316 \100121 \100175\324";
pltext(xy=1,3) = "Sm\277rebr\277d";
pltext(xy=4,3) = "pltext (xy=1,3) = \324Sm\\277rebr\\277d\324";
plotp (a=1,1) = 3.5,4.05,2.7,4.05;
plotp (a=1,1) = 3.5,3.05,2.7,3.05;
plotk (sc=-90) = 1.5,1.5,3.2,1.5,3.2,2.1;
plotp (a=1,1) = 3.2,2.1,2.9,2.1;
plotk (sc=-90) = 1.5,1.5,1.5,0.95;
plotp = 1.5,0.95,2,0.95;
```

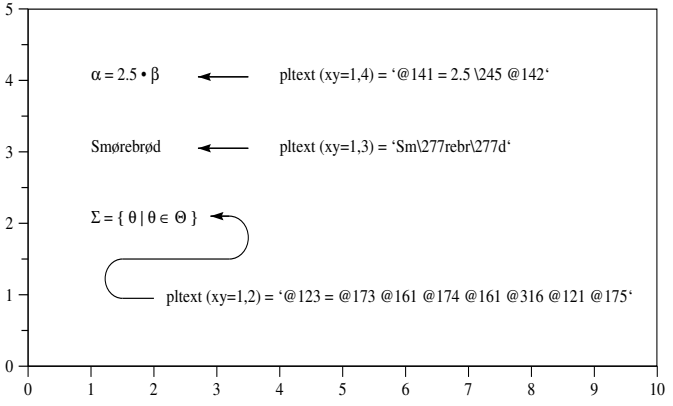


Figure 3 PostScript plot `plots.ps`, created with command file `plots.cf` shown in Box 1.

4.4.4 Rectangles

To plot a rectangle one can use the `plrec` command with syntax shown in the following box.

```
plrec (
     lt=...,          line type, def. 1 (solid line)
     lw=...,          line width, def. 0.2 (mm)
     gs=...,          grey scale value, def. 1 (white)
     r=...,           rotation, def. 0
) = x,y,xlen,ylen;
```

The command plots a rectangle with the lower left corner at the point `(x,y)`; `xlen` is the width and `ylen` the height of the rectangle (in logical coordinates). All other parameters are optional. Default line type is `lt=1`, default line width is `lw=0.2`. The `gs` parameter can be used to fill the rectangle with a grey tone. The `r` parameter can be used to rotate the rectangle. Note that `xlen` and `ylen` can be zero. Of course, the rectangle is not plotted then; but the bounding box is updated to include the point `(x,y)`.

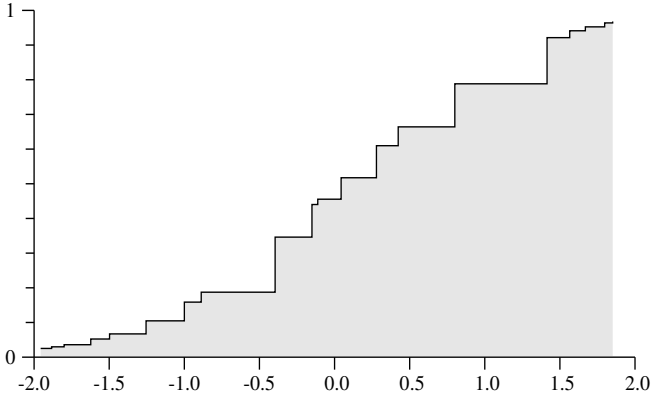
4.4.5 Step Functions

The `plot` and `plotp` commands can be used to plot step functions and histograms. The parameter `r=1` draws straight lines to connects the points of the polygon with the x axis. The parameter `dir=1` has the effect that the data points are connected by "kinked" line segments where the pen moves first in the x axis direction, then in the y axis direction. This creates a step function. Alternatively, with `dir=2`, the pen moves first in the y axis direction, then in the x axis direction. By default, the command draws horizontal and vertical lines. Adding the parameter `ns=1`, only horizontal lines will be drawn.

Example 1 To illustrate these options, we modify `plot8.cf`, the command file that was used in 4.4.1 to illustrate the `plot` command. In the new command file, `plot11.cf` (not shown), the plot command is

```
plot (dir=1,gs=0.9) = RDS,ND;
```

The resulting grey-scaled step function is shown in the following figure.



4.4.6 General Functions

The `plotf` command can be used to plot a function $y = f(x)$. The `plotf1` and `plotf2` commands will plot, respectively, the first and second derivative of the given function. All three commands use the same syntax as shown in the following box.

```
plotf (
     rx=...,        range of function argument
     lt=...,        line type, def. 1 (solid line)
     lw=...,        line width, def. 0.2 (mm)
     gs=...,        grey scale value, def. 1 (white)
     nc=...,        no clipping option, def. 0
) = function;
```

The right-hand side must provide the definition of a function, depending on a single argument, as explained in **5.3.1**. If the definition refers to at least one data matrix variable, the function is summed over all data matrix cases.

The function is evaluated in the range [a,b] on the x axis, with increments of size d; these values must be provided with the `rx=a(d)b` parameter. For each evaluation, the argument is used as the x axis coordinate and the value of the function as the y axis coordinate. The data points are connected by straight line segments. The line type and line width can be controlled with the `lt` and `lw` parameters, respectively. Defaults are `lt=1` (solid line) and `lw=0.2` (mm).

Optionally, one can use the `gs` parameter to fill the area below the function with a grey tone. The `nc=1` parameter can be used to allow a plot of the function outside the coordinate system.

Box 1 Command file `plot12.cf`

```
psfile = plot12.ps;  output file
psetup(
    pxlen =  90,
    pylen =  50,
    pxa   = -3,3,
    pya   =  0,1,
);
plxa (sc=0.5,fmt=4.1);  plot x axis
plya (sc=1,ic=10);      plot y axis
plframe;                plot frame

plotf (rx=0(0.05)3,) = ndf((x - 1.5) / 0.5) / 0.5;
plotp = 1.5,0,1.5,0.798;
plotf(rx=-3(0.1)3,gs=0.8) = ndf(x);
plotp = 0,0,0,0.399;
```

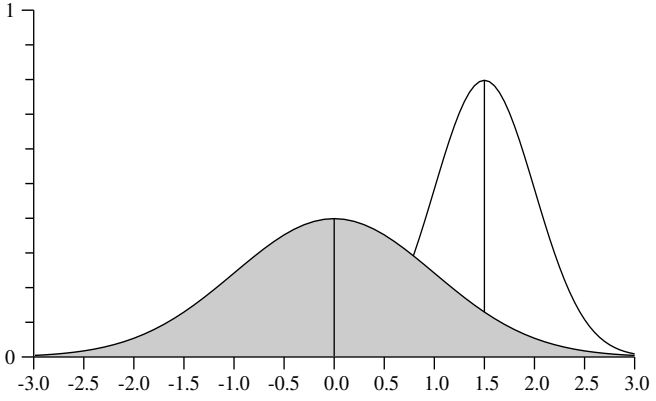


Figure 1 Example of a PostScript plot, created with command file `plot12.cf` shown in Box 1.

Example 1 To illustrate the `plotf` command, `plot12.cf` (Box 1) creates plots of two normal density functions, one grey-scaled, and adds two straight lines at the means. The result is shown in Figure 1. Note that when using grey scales, the effect depends on the ordering of commands since a grey-scaled object can mask previous objects.

4.4.7 Arrows

The `plot` and `plotp` commands (see 4.4.1) can be modified such that the resulting polygon ends in an arrow. The parameter is

```
a = xs,ys
```

with `xs` defining the length and `ys` the width of the arrow head (in mm), respectively.

Command file `plot13.cf`, shown in Box 1, uses the `plotp` command to illustrate several arrows with different line types and head sizes. The plot is shown in Figure 1.

Box 1 Command file `plot13.cf`

```
psfile = plot13.ps;  output file
psetup(
    pxlen =  90,
    pylen =  50,
    pxa   = 0,10,
    pya   =  0,6,
);
plxa (sc=1);    plot x axis
plya (sc=1);    plot y axis
plframe;        plot frame

plotp(a=1,1)           = 5,3,8,3;
plotp(a=1.5,1.5)       = 5,3,7,5;
plotp(a=2,2)           = 5,3,5,5.5;
plotp(a=2.5,2.5)       = 5,3,3,5;
plotp(a=1,1,lt=5)      = 5,3,2,3;       use line type 5
plotp(a=1.5,1.5,lt=5)  = 5,3,3,1;       use line type 5
plotp(a=2,2,lt=5)      = 5,3,5,0.5;     use line type 5
plotp(a=2.5,2.5,lt=5)  = 5,3,7,1;       use line type 5

pltext (xy=8.3,3)   = [1,1];            some labels
pltext (xy=7.3,5)   = [1.5,1.5];
pltext (xy=5.3,5.6) = [2,2];
pltext (xy=1.6,5)   = [2.5,2.5];
```

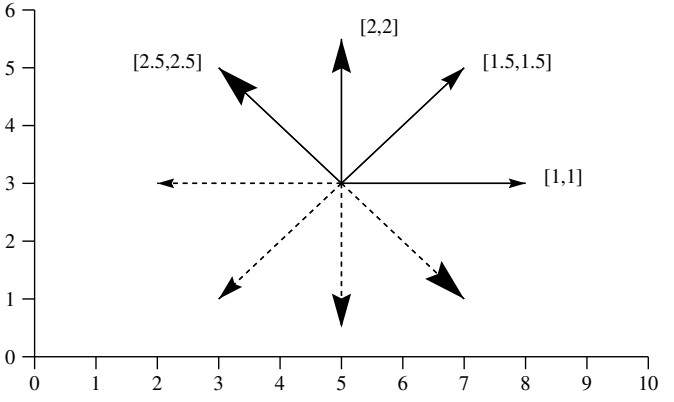


Figure 1 PostScript plot `plot13.ps`, created with command file `plot13.cf` shown in Box 1.

4.4.8 Circles, Arcs, Ellipses

The `ploto` command can be used to plot circles and arcs; for an illustration, see Figure 1. The syntax is shown in the following box.

```
ploto (
     xy=...,          coordinates of center
     lt=...,          line type, def. 1 (solid line)
     lw=...,          line width, def. 0.2 (mm)
     gs=...,          grey scale value, def. 1 (white)
     nc=...,          no clipping option, def. 0
) = r [,alpha,beta];
```

The `xy=x,y` parameter must be used to define the center of the circle. The radius must be given on the right-hand side. The angles, α and β, to be given in degrees, are optional. If these angles are not given, the result is a full circle, otherwise it is an arc, beginning at the angle α and ending at the angle β, in counterclockwise direction; default values are $\alpha = 0$ and $\beta = 360$ to provide a full circle. Other parameters are optional. `lt` and `lw` can be used to change the default line type and line width. `gs` can be used to fill the circle with a grey tone. And `nc=1` can be used to plot outside the coordinate system.

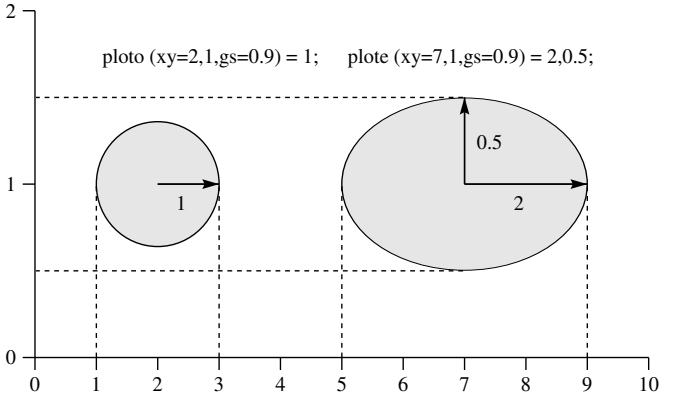


Figure 1 PostScript plot `plot15.ps`, created with command file `plot15.cf` (not shown), to illustrate the `ploto` and `plote` commands.

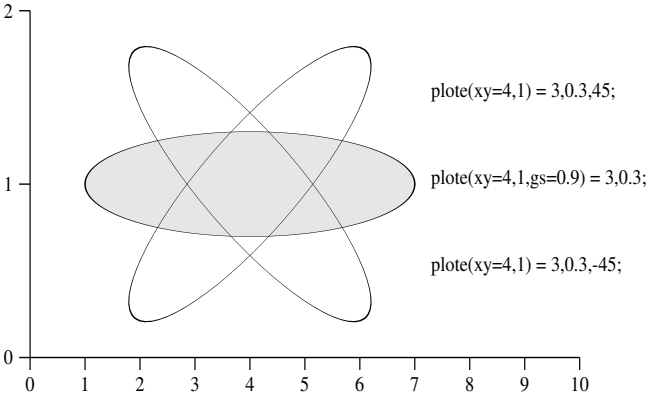


Figure 2 PostScript plot `plot16.ps`, created with command file `plot16.cf` (not shown), to illustrate the `ploto` and `plote` commands.

The `plote` command can be used to plot ellipses; for an illustration see Figure 2. The syntax is shown in the following box.

```
plote(
     xy=...,        coordinates of center
     lt=...,        line type, def. 1 (solid line)
     lw=...,        line width, def. 0.2 (mm)
     gs=...,        grey scale value, def. 1 (white)
     nc=...,        no clipping option, def. 0
) = a,b [,r];
```

The `xy=x,y` parameter must be used to define the center of the ellipse. `a` is the half length of the main axis (interpreted in user coordinates of the x axis), `b` is the half length of the second axis of the ellipse (interpreted in user coordinates of the y axis). `r` is optional; if provided the ellipse is rotated by `r` degrees.

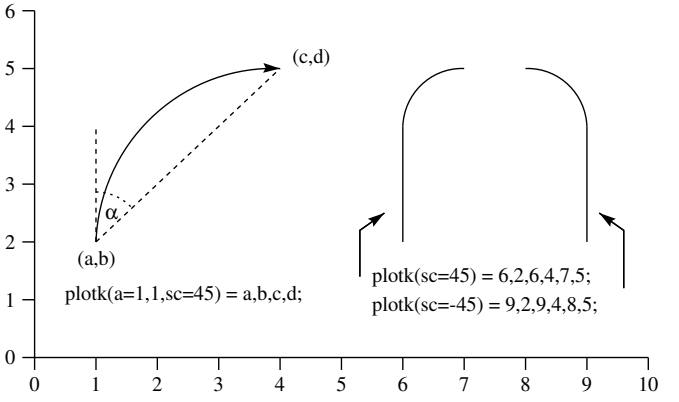


Figure 3 PostScript plot `plot14.ps`, created with command file `plot14.cf` (not shown), to illustrate the `ploto` and `plotk` commands.

To plot arcs which optionally end in an arrow, one can use the `plotk` command; for an illustration see Figure 3. The syntax is shown in the following box.

```
plotk(
     sc=...,          degree of curvature, def. 0
     lt=...,          line type, def. 1 (solid line)
     lw=...,          line width, def. 0.2 (mm)
     a=...,           size of arrow head
     nc=...,          no clipping option, def. 0
) = x1,y1,...,xn,yn;
```

Given this command, the points $(x_1, y_1), \ldots, (x_{n-1}, y_{n-1})$ are connected by a polygon, and the points (x_{n-1}, y_{n-1}) and (x_n, y_n) are connected by an arc. The arc is constructed so that its radius is defined implicitly by the angle α to be given in degrees with the parameter `sc=alpha`; see the example given in Figure 3. Note that α is restricted to the range $-180 < \alpha < 180$.

As an additional option one can use the `a=xs,ys` parameter. The polygon then ends in an arrow with `xs` defining the length and `ys` the width of the arrow head (in mm), respectively.

4.4.9 Convex Hull

To plot the convex hull of a set of points, TDA uses an algorithm published by Eddy [1977]. The command is `plotch` with syntax shown in the following box.

```
plotch (
     lt=...,        line type, def. 1 (solid line)
     lw=...,        line width, def. 0.2 (mm)
     s=...,         marker symbol
     fs=...,        size of marker symbol, def. 2 (mm)
     gs=...,        grey scale value, def. 1 (white)
     nc=...,        no clipping option, def. 0
     ns=...,        number of intervals for smoothing
     ic=...,        added to smoothed hull, def. 0
     sel=...,       selection of data points
) = VX,VY;
```

The right-hand side of the command must contain exactly two variable names, to be used for the x and y coordinates of the data points. All other parameters are optional.

1. By default, the command uses all cases (rows) of the currently defined data matrix. To select a subset of cases one can use the `sel` parameter, see the description of the `plot` command in 4.4.1.

2. The `lt` and `lw` parameters can be used to modify the default line type and line width.

3. By default, only the convex hull polygon is plotted. To plot the data points, one can provide the number of a marker symbol with the `s` parameter. The `fs` parameter can then be used to change the default size of the symbols.

4. The `gs` parameter can be used to fill the convex hull with a grey tone. The `nc=1` (no clipping) parameter can be used to allow plotting outside of the coordinate system.

5. Finally, one can request a smoothing of the convex hull with the `ns`

Box 1 Command file `plot17.cf`

```
nvar(
    noc = 100,
    X = rd,
    Y = rd,
);
psfile = plot17.ps;  output file
psetup(
    pxlen =  90,
    pylen =  50,
    pxa = -0.5,1.5,
    pya = -0.5,1.5,
);
plxa (sc=0.5);
plya (sc=0.5);
plframe;
plotch(gs=0.95,s=5,fs=0.7) = X,Y;
```

parameter. TDA uses then the Akima algorithm, described in 4.4.10, to create a smoothed version of the convex hull polygon. (Note that the resulting graph is no longer a convex hull in its strict definition.) The parameter must be given as `ns=n` with `n` an integer greater than, or equal to, 2. This specifies the number of subintervals used for smoothing. As an additional option, one can use the `ic=x` parameter with some value `x > 0`. Depending on the value of `x`, this results in some blow up of the smoothed convex hull in such a way that all data points are strictly inside its area.

Example 1 Command file `plot17.cf`, shown in Box 1, illustrates the `plotch` command. In a first step, the command file creates 100 random data points (using the `rd` operator). The `plotch` command is then used to plot the convex hull of these data points. The area is grey-scaled with value `gs=0.9`. A scatterplot of the data points is added with the `s` parameter. The resulting plot is shown in Figure 1.

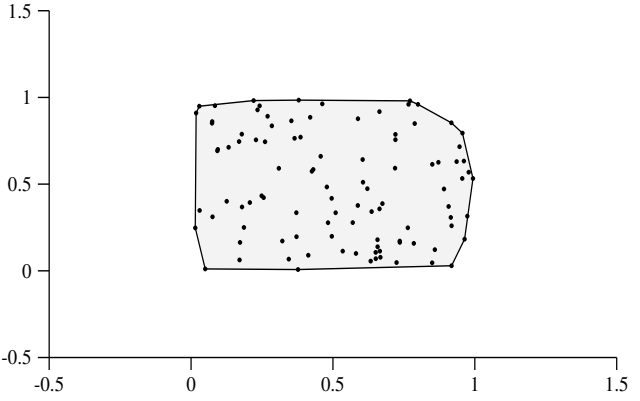


Figure 1 PostScript plot `plot17.ps`, created with command file `plot17.cf` shown in Box 1.

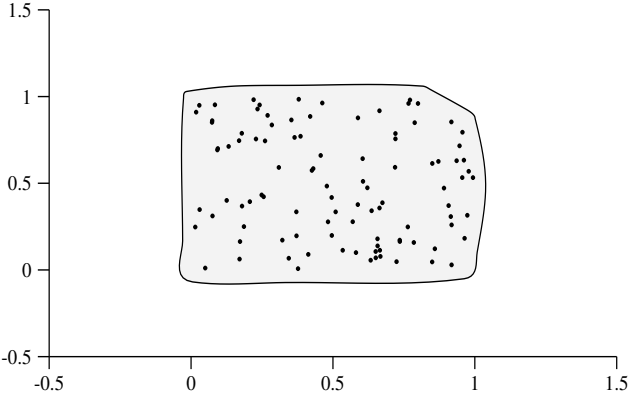


Figure 2 PostScript plot `plot18.ps`, created with command file `plot18.cf` (not shown), to illustrate a smoothed convex hull plot.

Example 2 To illustrate the smoothing option, we have changed the `plotch` command into

```
plotch(ns=10,ic=2,gs=0.95,s=5,fs=0.7) = X,Y;
```

The new command file, `plot18.cf`, is not shown here, but the resulting plot is shown in Figure 2. The blow up parameter is `ic=2` (mm).

4.4.10 Smoothing Polygons

For smoothing polygons TDA uses an algorithm published by Akima [1972]. There are two commands. The command `plots`, with syntax shown in the following box,

```
plots(
     ns=...,        degree of smoothing, def. 2
     lt=...,        line type, def. 1 (solid line)
     lw=...,        line width, def. 0.2 (mm)
     s=...,         marker symbol
     fs=...,        size of marker symbol, def. 2 (mm)
     gs=...,        grey scale value, def. 1 (white)
     nc=...,        no clipping option, def. 0
     sc=...,        request closed curve (sc=1), def. sc=0
     df=...,        write data to output file
     fmt=...,       print format
     sel=...,       selection of data points
) = VX,VY;
```

can be used analogously to the `plot` command, see 4.4.1. The right-hand side must provide the names of two variables containing the coordinates of the data points.

1. The `sel` parameter can be used to select a subset of data points.

2. The degree of smoothing can be controlled with the `ns` parameter. The syntax is `ns=n` with `n` an integer greater than, or equal to, 2; default is `ns=2`.

3. The `sc=1` parameter can be used to treat the polygon as a closed curve. The first and last point of the polygon are then connected before the smoothing algorithm is applied. These points are be marked by symbols if the `s` parameter has been used to specify the number of a marker symbol; the `fs` parameter can then be used to change the default symbol size.

4. The `gs` parameter can be used to fill the smoothed curve with a grey tone. The effect depends on the `sc` parameter. By default, the area

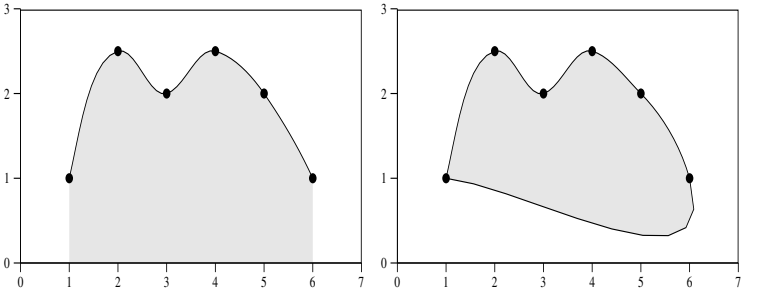


Figure 1 PostScript plots created with command files `plot19.cf` and `plot20.cf` to illustrate smoothed polygons. In the right plot, the parameter `sc=1` has been used to request a closed curve.

between the smoothed curve and the x axis is grey-scaled. With the `sc=1` parameter, the area inside the closed curve is grey-scaled.

5. The `lt` and `lw` parameters can be used to change the default line type and line width. The `nc=1` (no clipping) parameter allows a plot outside the area of the coordinate system.

5. Finally, one can provide the name of an output file with the `df` parameter. Coordinates of the smoothed data points will then be written into this file. The print format can be controlled with the `fmt` parameter.

Alternatively, one can use the command

```
plotsp (ns=,...) = x1,y1,x2,y2,...,xn,yn;
```

that allows to provide a list of data points directly on the right-hand side of the command. All other parameters have the same meaning as explained above for the `plots` command.

Example 1 An example, based on some arbitrarily chosen data points, is shown in Figure 1. The command for the plot on the right side is

```
plotsp(ns=10,gs=0.9) = 1,1,2,2.5,3,2,4,2.5,5,2,6,1;
```

For the plot on the right side the parameter `sc=1` was added to request a closed curve.

4.4.11 Contour Plots

To get information about a single-valued function with two arguments, $f(x, y)$, it is sometimes useful to plot its projection onto a two-dimensional plane. This is called a *contour plot*. To provide this option, TDA uses an algorithm published by Snyder [1978]. The command to request a contour plot is `plotc` with syntax shown in the following box.

```
plotc (
     x=...,          specification of levels
     nn=...,         grid specification, def. nn=10,10
     lt=...,         line type, def. 1 (solid line)
     lw=...,         line width, def. 0.2 (mm)
     gs=...,         grey scale value, def. 1 (white)
) = function;
```

The right-hand side must contain the definition of a function depending on (up to) two parameters, specified by the key words `x1` (for the x axis) and `x2` (for the y axis). Depending on these arguments, the function can be defined as any valid expression, which may contain numerical constants, random numbers, and any of TDA's type 1 operators. It must be possible to evaluate the function over the whole range of the given coordinate system.

The function is evaluated for the node points of a grid on the coordinate system. The grid is defined by integers `nx` and `ny`, the number of intervals on the x and y axis, respectively. The parameter to provide these numbers is `nn=nx,ny`; default is `nn=10,10`.

In addition, the command requires the specification of a sequence of levels with the `x=l1,l2,...` parameter. For each of these levels the algorithm tries to plot a contour line defined by the curve $f(x, y) = l_i$.

The other parameters are optional. `lt` and `lw` can be used to modify the default line type and line width. The `gs` parameter can be used to request a grey-scaled contour plot. The levels should then be ordered from lowest to highest. Given the parameter `gs=g`, each level gets a

Box 1 Command file `plot21.cf`

```
psfile = plot21.ps;  output file
psetup(
    pxlen =  70,
    pylen =  70,
    pxa = -2,2,
    pya = -2,2,
);
plxa (sc=1);
plya (sc=1);
plframe;

plotc (x=0.01,0.05,0.1,0.125,0.15,nn=50,50,gs=0.9) =
    exp(-((x1 + 1)^2 + (x2 + 1)^2) / 2) / (2 * pi) +
    exp(-((x1 - 1)^2 + (x2 - 1)^2) / 2) / (2 * pi);
```

grey-scale value according to

$$\text{greyscale value} = g\left(1 + \frac{1-i}{n}\right) \qquad i = 1, \ldots, n$$

One should note that, to save programming overhead, only *closed* contour lines are filled with grey-scaled shades. However, a contour plot is just another plot object, so it is possible to have many contour plots and other plot objects in the same picture. In particular, the bounding box of the coordinate system can be given a greyscale before adding contour plots.

Example 1 To illustrate the contour plot command, our function is a mixture of two two-dimensional normal densities with zero correlation, defined symmetrically around zero:

$$f(x_1, x_2) = \sum_{i=1}^{2} \frac{1}{2\pi} \exp\left(-\frac{1}{2}\left[(x_1 - \mu_i)^2 + (x_2 - \mu_i)^2\right]\right)$$

The means are selected to be $\mu_1 = -1$ and $\mu_2 = 1$. Box 1 shows the command file, `plot21.cf`, to create a contour plot for this function, using five levels in ascending order. Figure 1 shows the resulting plot.

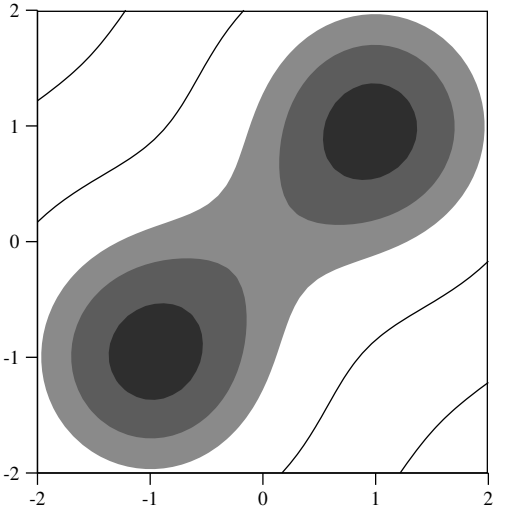


Figure 1 PostScript plot `plot21.ps`, created with command file `plot21.cf` shown in Box 1.

4.5 Built-in Previewer

In order to preview PostScript plots, one most often will use a general purpose preview utility, e.g., GhostScript or GhostView. For very simple plots one can also use TDA's built-in previewer. This option is available when TDA runs in a Windows environment. It relies on a set of graphic primitives for an X-Windows environment, and the previewer will therefore work properly only when TDA runs in an X-Windows environment of a UNIX system. There are, however, some substitutes when TDA runs in an MS-Windows environment. In any case, the previewer is not available for a DOS version of TDA.

This section describes four commands that can be used to work "interactively" with TDA PostScript files. Some additional ready-made plot commands will be described in 4.6.

1. The command

   ```
   xopen = psfile;
   ```

   requires that `psfile` is the name of a PostScript file that was created by TDA. The command then makes this file TDA's *current* PostScript file and one can add and delete plot objects.

2. The command

   ```
   xlog [= psfile];
   ```

   shows the plot objects in TDA's current PostScript file or, alternatively, in the file specified on the right-hand side. Each plot object is shown by a number and a short description in TDA's standard output.

3. The command

   ```
   xdelete = n1,n2,... ;
   ```

   deletes the plot objects with numbers `n1,n2,...` from the current PostScript file.

4. The command

   ```
   xshow [= psfile];
   ```

invokes the built-in previewer to show a picture of the current PostScript file or, alternatively, the PostScript file specified on the right-hand side, on the screen. If possible, the previewer opens a plot window and shows the plot in this window. Note that TDA remains inactive as long as the plot window is open. Note also that, in an MS Windows environment, one has to "click" on the window in order to get the picture.

As an option, the command can also be used as

```
xshow (dscale=...) [= psfile];
```

where the `dscale` parameter provides a scaling factor for the plot. The default scaling factor is 1.

Limitations. One should note that the built-in previewer has a number of limitations. Each single plot object may only contain up to 1000 data points; and the previewer can not show text and grey-scaled areas.

4.6 Ready-made Plot Commands

There are a few ready-made plot commands that have the following features:

- If there is no current open PostScript file, they create a new PostScript file and automatically create a coordinate system in such a way that the new plot object roughly fits. A new PostScript plot file is always named `xplot.ps`.
- If a current PostScript file exists, these commands (like all other plot commands) add a plot object to this file.

The main usage of these commands is to allow for "quick and dirty" plots that can be viewed with the built-in previewer (see 4.5). The following pages describe the currently available commands.

Box 1 Syntax for `xplot` command

```
xplot (
     opt=...,       plot option, def. 1
                    1 : scatterplot
                    2 : line plot
     s=...,         symbol type, def. 1
     fs=...,        symbol size, def. 1.3 mm
     lt=...,        line type, def. 1
     lw=...,        line width, def. 0.2 mm
     pxlen=...,     length of x axis, def 120 mm
     pylen=...,     length of y axis, def 80 mm
) = VX,VY [,VG];
```

Standard Line and Scatter Plots

By default, the `xplot` command creates a scatterplot based on the data in the first two variables given on the right-hand side. As an option (`opt=2`) the data points are connected by straight lines. As a further option, one can add a third variable on the right-hand side. If added, each set of data points having the same value of this variable is treated as a separate group and is shown by a separate plot symbol. Note that the `s` parameter can only be used when there is only one group.

Box 2 Syntax for `xplotf` command

```
xplotf (
     cn=...,          column numbers of variables, def. cn=1,2
     gn=...,          column number of group variable
     opt=...,         plot option, def. 1
                      1 : scatterplot
                      2 : line plot
     s=...,           symbol type, def. 1
     fs=...,          symbol size, def. 1.3 mm
     lt=...,          line type, def. 1
     lw=...,          line width, def. 0.2 mm
     pxlen=...,       length of x axis, def 120 mm
     pylen=...,       length of y axis, def 80 mm
) = name_of_data_file;
```

Standard Line and Scatter Plots Based on a Data File

This command works similar to the `xplot` command but gets its data directly from a free format data file that must be specified on the right-hand side.

a) By default, the command uses the first two data columns in the input file to get values for the X and Y variables.

b) Optionally, one can use the `cn` parameter to specify data column numbers: `cn=ci,cj,ck,...` This specifies the data point groups:

```
(Xi,Xj), (Xi,Xk), ...
```

and a scatterplot, or line plot, is created separately for each group.

c) As a further option, one can specify

```
cn=ci,cj, gn=ck,
```

in order to define corresponding variables `VX`, `VY`, and `VG`. `VX` and `VY` provide the data points, and `VG` is a grouping variable, meaning that each set of `(VX,VY)` values which have the same value in `VG` is treated as a separate group.

Box 3 Syntax for `xf` command

```
xf (
     rx=...,        definition of x axis, def. rx=1(1)10
     lt=...,        line type, def. 1
     lw=...,        line width, def. 0.2 mm
     pxlen=...,     length of x axis, def 120 mm
     pylen=...,     length of y axis, def 80 mm
) = function(x);
```

Functions

This command plots `function(x)`. If there is a currently open PostScript file, the function is plotted for the already defined x axis. Otherwise, the command creates a new PostScript file. By default, the function is then plotted for an x axis from 1 to 10. Alternatively, one can define the x axis with the parameter

```
rx = a (d) b,
```

The range is then from `a` to `b` with increments `d`.

Box 4 Syntax for `xconh` command

```
xconh(
     lt=...,        line type, def. 1
     lw=...,        line width, def. 0.2 mm
) = function(x);
```

Convex Hulls

This command adds a convex hull to the most recent scatterplot. If the scatterplot consists of more than one group, the convex hulls are created separately for each group.

Box 5 Syntax for `xreg` command

```
xreg(
     sig=...,        sigma for lowess, def. 0.5
     lt=...,         line type, def. 1
     lw=...,         line width, def. 0.2 mm
) [= n];
```

Regression Curves

This command adds a regression curve to the most recent scatterplot. In case of more than one group of points this is done separately for each group. One of there options can be selected on the right-hand side:

`1` Lowess regression curve (default)

`2` Least squares regression line

`3` L1 norm regression line

4.7 3D Coordinate Systems

This chapter describes how to define and use three-dimensional coordinate systems for PostScript plots.

4.7.1 Coordinates and Projections

In order to create 3-dimensional PostScript plots it is necessary to define a PostScript output file with the `psfile` command and to set up a coordinate system and projection with the `psetup3` command:

4.7.2 The `plot3` and `plotp3` Commands

There are two commands to plot an arbitrary sequence of data points, `plot` and `plotp`. The only difference is in the way the data points are supplied. With the former one, the data points are taken from the program's internal data matrix, with the latter one, the data points can be explicitly supplied as part of the command.

4.7.3 Plotting Text

There are two commands to plot an arbitrary sequence of data points, `plot` and `plotp`. The only difference is in the way the data points are supplied. With the former one, the data points are taken from the program's internal data matrix, with the latter one, the data points can be explicitly supplied as part of the command.

4.7.4 Parametric Curves

There are two commands to plot an arbitrary sequence of data points, `plot` and `plotp`. The only difference is in the way the data points are supplied. With the former one, the data points are taken from the program's internal data matrix, with the latter one, the data points can be explicitly supplied as part of the command.

4.7.5 Parametric Surfaces

There are two commands to plot an arbitrary sequence of data points, `plot` and `plotp`. The only difference is in the way the data points are supplied. With the former one, the data points are taken from the program's internal data matrix, with the latter one, the data points can be explicitly supplied as part of the command.

4.7.6 Arbitrarily Distributed Points

There are two commands to plot an arbitrary sequence of data points, `plot` and `plotp`. The only difference is in the way the data points are supplied. With the former one, the data points are taken from the program's internal data matrix, with the latter one, the data points can be explicitly supplied as part of the command.

5. Mathematical Procedures

This part contains the following sections.

5.1 Working with Matrices

This chapter explains how to use TDA for matrix calculations. It contains the following sections.

5.1.1 Introduction and Overview

5.1.2 Matrices explains the basic concepts and how to create and print matrices.

5.1.3 Matrix Expressions explains TDA's concept of matrix expressions.

5.1.4 Matrix Commands explains all currently available matrix commands.

5.1.1 Introduction and Overview

Command		Section
mdef	creates a matrix	5.1.2
mdefb	creates a matrix from a block	5.1.2
mdeff	reads a matrix from a file	5.1.2
mdefi	creates an identity matrix	5.1.2
mdefc	creates a matrix with constant values	5.1.2
mdefg	creates a graph's adjacency matrix	5.1.2
mfmt	changes the print format	5.1.2
mfree	deletes matrices	5.1.2
mnvar	creates internal data matrix	5.1.2
mpr	prints a matrix/expression	5.1.2
mpra	appends a matrix/expression	5.1.2
mexpr	evaluates a matrix expression	5.1.3
mexpr1	evaluates a matrix expression	5.1.3
msetv	changes elements of a matrix	5.1.3
mag	matrix aggregation	5.1.4.1
mcath	horizontal concatenation	5.1.4.1
mcathv	direct sum	5.1.4.1
mcatv	vertical concatenation	5.1.4.1
mcel	edge list from adjacency matrix	5.1.4.14
mcent	centering	5.1.4.3
mchol	Cholesky decomposition	5.1.4.6
mcross	cross product	5.1.4.1
mcsum	column sums	5.1.4.1
mcvec	stacks columns	5.1.4.1
mdcent	double centering	5.1.4.3
mdcol	creates diagonal matrix	5.1.4.1
mdiag	creates diagonal matrix	5.1.4.1
mdiagd	extracts diagonal	5.1.4.1
mdrow	creates diagonal matrix	5.1.4.1
mev	eigenvalues and eigenvectors	5.1.4.11
mevs	eigenvalues of symmetric matrix	5.1.4.11
mginv	generalized inverse	5.1.4.5
minvd	inverse of diagonal matrix	5.1.4.5
minvs	inverse of symmetric matrix	5.1.4.5

Command		Section
mkp	Kronecker product	5.1.4.1
mlp	linear programming	5.1.4.13
mlp1	linear programming with constraints	5.1.4.13
mls	least squares regression	5.1.4.7
mlse	linear equations	5.1.4.8
mlsei	regression with constraints	5.1.4.10
mlsi	linear inequalities	5.1.4.9
mmul	multiplication	5.1.4.1
mncol	extracts number of columns	5.1.3
mnls	regression with constraints	5.1.4.10
mnorm	maximal norm	5.1.4.4
mnorm1	L1 norm	5.1.4.4
mnorm2	Euclidean norm	5.1.4.4
mnrow	extracts number of rows	5.1.3
mpbl	lower block diagonal matrix	5.1.4.14
mpbu	upper block diagonal matrix	5.1.4.14
mpcol	column permutation	5.1.4.1
mpfit	iterative proportional fitting	5.1.4.14
mpinv	inverse permutation	5.1.4.1
mpit	Leslie matrix iteration	5.1.4.14
mple	Kaplan-Meier estimates	5.1.4.14
mproc	Procrustes rotation	5.1.4.14
mprow	row permutation	5.1.4.1
mpsym	symmetrical permutation	5.1.4.1
mpz	zero-free main diagonal	5.1.4.14
mqap	quadratic assignment	5.1.4.14
mrank	ranking	5.1.4.2
mrsum	row sums	5.1.4.1
mrvec	stacking rows	5.1.4.1
mscal1	scaling	5.1.4.3
mscol	selection of columns	5.1.4.1
msort	sorting	5.1.4.2
msort1	sorting	5.1.4.2
msqrtd	square root of diagonal matrix	5.1.4.1
msqrti	inverse square root of diagonal matrix	5.1.4.1
msrow	selection of rows	5.1.4.1
mstand	standardization	5.1.4.3
msvd	singular value decomposition	5.1.4.12
msvd1	singular value decomposition	5.1.4.12
mtrace	trace	5.1.4.1
mtransp	transposition	5.1.4.1
mwvec	construction of weights	5.1.4.14
mwvec1	construction of weights	5.1.4.14

5.1.2 Matrices

Matrix Names

Matrices are rectangular arrays of numerical entries. A general (m, n) matrix X, that is, a matrix with m rows and n columns, is of the form

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$$

X is the name of the matrix. TDA uses the same conventions for matrix and variable names (see 2.1). A matrix name is an arbitrary string of up to 32 characters which may consist of the following letters:

A,...,Z,a,...,z,0,1,...,9,_,@,$

The first character of a matrix name must be an upper case letter, or one of the characters "_", "@", "$". Note that matrix names are case sensitive, that is, lower and upper case letters are distinguished. There is no special notation for vectors. Column vectors are treated as $(m, 1)$ matrices, row vectors as $(1, n)$ matrices.

Storage of Matrices

Matrices are always stored in double precision (8 byte) arrays. To store an (m, n) matrix needs $(m \cdot n + 1) \cdot 8$ bytes. There is an upper limit of `MaxMat` matrices. When invoking the program, the default (`MaxMatDef = 200`) can be changed with the `maxmat` command (see 1.4). Memory to store matrices is only allocated when a new matrix becomes defined, and if a matrix is deleted, the previously allocated memory will be given back to the operating system and can be used then for other purposes.

Creating Matrices

There are several different options to create matrices.

1. The command

   ```
   mdef(X);
   ```

   creates a (m, n)-matrix `X` with m equal to the number of rows in the current internal data matrix and n equal to the current number of variables. The command then copies the internal data matrix into `X`.

2. The command

   ```
   mdef(X,m,n);
   ```

   creates a (m, n)-matrix `X` and copies a maximum of m rows and n columns (variables) from the internal data matrix into `X`.

3. The command

   ```
   mdef(X) = varlist;
   ```

   creates a (m, n)-matrix `X` with m equal to the number of rows in the current internal data matrix and n equal to the number of variables in `varlist`. The command then copies the values of the specified variables into `X`.

4. The command

   ```
   mdef(X,m,n) = varlist;
   ```

   creates a (m, n)-matrix `X` and copies the values for maximal n variables from `varlist` and for maximal m rows in the current data matrix into `X`. If m is greater then the number of cases (`noc`) or if n is greater than the number of variables in `varlist`, the corresponding rows or columns are filled with zeros.

5. The command

   `mdef(X,m,n) =` $x_{11}, \ldots, x_{1,n}, \ldots, x_{m1}, \ldots, x_{m,n}$`;`

   creates a (m, n)-matrix `X` and uses for its elements the values given on the right-hand side. See Box 1 for an example.

6. The command

   ```
   mdef(X,m,n) = file_name;
   ```

creates a (m, n)-matrix `X`. The command then tries to open a data file with the name given on the right-hand side and reads a maximum of m data records from the file. From each record, the command uses the first n numerical entries for the elements of `X`. It is assumed that the file is organized as a free-format file.

7. The command

```
mdeff(X) = file_name;
```

tries to open a free-format data file with the name specified on the right-hand side. Given that the file contains m data records and n numerical entries in each record, the command then creates a (m, n)-matrix `X` and copies the numerical entries from the file into the elements of `X`.

8. Given a relational data structure, the command

```
mdefg(
    gn=...,
    sc=...,
) = matrix_name;
```

creates a matrix containing the graph's adjacency matrix. Except for the name of a matrix to be given on the right-hand side all parameters are optional. The `gn` parameter can be used to select a graph from the current relational data structure. By default, the command uses the first graph. The `sc` parameter can be used to substitute specific values for missing edges.

Special Matrices

1. The command

```
mdefc(m,n,x,A);
```

creates a (m, n)-matrix `A` and sets all of its elements to the value `x`. `x` must be provided as a scalar (matrix) expression.

2. The command

```
mdefi(m,n,A);
```

creates a (m, n)-matrix `A` and sets $\mathtt{A}(i, i) = 1$, for $i = 1, \ldots, \min(m, n)$.

Box 1 Illustration of `mdef` command

```
# command file: mat1.cf

mdef(I,3,3) = 1,0,0,
              0,1,0,
              0,0,1;
mpr(I);

Standard output
---------------
mdef(I,3,3)=...
mpr(I)
    1.0000     0.0000     0.0000
    0.0000     1.0000     0.0000
    0.0000     0.0000     1.0000
```

3. The command

   ```
   mnum(x,d,m,A);
   ```

   assumes that `x`, `d`, and `m` are scalar (matrix) expressions. The command creates a $(m, 1)$-matrix `A` and sets

   $$\texttt{A}(i) = \texttt{x} + (i-1)\texttt{d} \qquad (i = 1, \ldots, \texttt{m})$$

   x, d, and m can be (1,1) matrices.

Exchange with Internal Data Matrix

As has been described above, part or the whole of TDA's internal data matrix can be used to create matrices. On the other hand, one can also use a matrix to create a new internal data matrix. The command is

```
mnvar(A);
```

where `A` is a (m, n) matrix. The command creates a new internal data matrix with m rows and n variables. Variable names are created by using the matrix name and adding column numbers. If a data matrix already exists the command exits with an error message.

Saving Data Matrix Blocks in a Matrix

The comamnd

```
mdefb(B,expression);
```

can be used to save a block of data matrix rows in a matrix. The command requires that the `dblock` command has been used previously to define data matrix blocks (see 1.9). `B` must be a valid matrix name, and `expression` must be a scalar expression that evaluates to a valid block number, say n. The `mdefb` command then creates a matrix named `B` and copies block number n from the current data matrix into this matrix. Block numbers can be written into a matrix using the `dblock` command, see Section 1.9.

Information about Matrices

The command `mdef`, without parameters, can be used to get information about the currently defined matrices. A table will be displayed containing, for each matrix, its name, its dimensions, and the command that was used to define or create the matrix.

Printing Matrices

The command to print matrices has syntax

```
mpr(X [,string]);
```

where `X` is the name of a matrix. This matrix is then written into the program's standard output using the currently defined print format for matrices. The print format can be changed with the command

```
mfmt = formatstring;
```

The default print format is `mfmt=10.4`. Alternatively, using the command

```
mpr(X [,string]) = name_of_an_output_file;
```

the matrix is written into the specified output file. Using `mpra` instead of `mpr` appends to the output file.

Deleting Matrices

The command

```
mfree(X);
```

can be used to delete the matrix `X` from the list of currently defined matrices. The same command without an argument, simply `mfree`, deletes all currently defined matrices. The memory used to hold the matrix data is then deallocated and can be used for new matrices or other purposes.

5.1.3 Matrix Expressions

Matrix expressions are a generalization of standard (scalar) expressions that are described in 5.2. The extension is that at all places where a standard expression may contain numerical constants, a matrix expression may contain the name of a matrix, a variable, or a namelist. The basic idea can best be explained by referring to the `mexpr` command that is used to evaluate matrix expression. The general syntax is

```
mexpr(op(A,...,V,...,L,...), R);
```

where `A,...` refers to matrix names, `V` refers to variable names, `L` refers to namelist names, and `op` is used to refer to some (combination of) type 1 operators. Data matrix variables are treated as $(m, 1)$ matrices with m the number of rows in the current data matrix. Namelists are treated as (m, n) matrices with m the number of rows in the current data matrix and n the number of variables in the namelist. Consequently, each of the referred objects, `A,...`, `V,...`, and `L,...`, can be considered as a matrix with dimensions, say, `row(`i`)` and `col(`i`)`, for the ith object in the matrix expression. The resulting matrix `R` then gets the dimensions

$$\max\{\texttt{row}(i)\} \times \max\{\texttt{col}(i)\}$$

If one of the objects (matrices, variables, namelists) has smaller dimensions, its missing rows and columns are created by cyclically using the available ones. For example, if a matrix `A` has n columns, its $(n+1)$th column will equal its first column, and so on; analogously for rows.

We now give some additional hints on the usage of matrix expressions.

1. Names of matrices, variables and namelists must be different.
2. There are two type 1 operators which may help when using matrix expressions. The operator

   ```
   row(expression)
   ```

   returns the row dimension of `expression`, and the operator

   ```
   col(expression)
   ```

returns the column dimension of `expression`.[1]

3. The evaluation of matrix expressions is performed element-wise. For example, given matrices

$$\mathtt{A} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad \mathtt{B} = (1, 2)$$

the result of the matrix expression

```
mexpr(A * B, R);
```

would be

$$\mathtt{R} = \begin{pmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{pmatrix}$$

4. Matrix expressions may contain type 2 operators. Not allowed are the `pre` and `suc` operators (but the `lag` operator will work), and the operators for episode and sequence data. When a matrix expression contains type 2 operators, the resulting matrix is calculated as a sequence of column vectors, meaning that the operators apply separately for each of these column vectors. For example,

```
mexpr(sort(X),R);
```

would separately sort each column of `X` in ascending order.

5. In order to allow the block mode operators for matrix expressions, there is an additional command,

```
mexpr1(B, expression, R);
```

where `B` is a (m, n) matrix and `expression` is some (r,c) matrix expression. It is required that $m \geq r$. The resulting matrix, `R`, has dimension $\max(m, r) \times c$ and is evaluated from `expression`. While evaluating the expression, `B` is used for the definition of blocks, that is, each consecutive number of identical rows in `B` is treated as a separate block.

[1] These are operators; the corresponding commands are `mnrow` and `mncol`.

6. A matrix expression may contain arguments

`X(row,col)`	where `X` is a matrix, or
`V(row,1)`	where `V` is a data matrix variable, or
`L(row,col)`	where `L` is a namelist.

   The result is the corresponding element of `X`, `V`, or `L`. For example,

```
mexpr(A(row,col),R);
```

   would create a scalar matrix, `R`, containing the value of `A(row,col)`. `row` and `col` can be any scalar expression.

7. Values resulting from `row` and `col` expressions must be positive integers. If they exceed the corresponding matrix dimensions, they are cyclically turned around. For example, if the matrix is

   `A` $= (1, 2, 3)$,

   then `A(1,5)` $= 2$.

8. In addition, one can use the following constructs (where `X` is a matrix, a variable, or a namelist):

`X(row,.)`	sum of the elements in the row
`X(.,col)`	sum of the elements in the column
`X(.,.)`	sum of all elements

9. All constructs mentioned so far are considered as type 1 operators and get the dimension $(1, 1)$. There is a further type 2 operator,

   `X`(i)

   which returns the ith column of the matrix (or variable, or namelist) `X`. The returned object has dimension `row`$\times 1$ where `row` is the number of rows of object `X`.

10. Whenever the argument of a matrix command refers to an existing matrix, it is possible to use an arbitrary matrix expression instead of a single matrix name. So it is possible to combine scalars, matrices, variables and namelists in all matrix commands. For example,

```
mpr(sin(V));
```

   where `V` is a data matrix variable would print the resulting vector.

11. As a further extension, one can directly refer to vectors with the syntax

    `< e1,e2,... >` for column vectors
    `< e1,e2,... >'` for row vectors

    where `e1, e2, ...` are any scalar expressions. For example,

    ```
    mpr(<1,2,3>');
    ```

    would be printed as the row vector $(1, 2, 3)$ and

    ```
    mpr(<1,2,3> + < 4,5,6 >');
    ```

    would be printed as the matrix

    $$\begin{pmatrix} 5 & 6 & 7 \\ 6 & 7 & 8 \\ 7 & 8 & 9 \end{pmatrix}$$

    If a vector has fewer elements as required by the dimension of its context, it is cyclically updated. For example,

    ```
    mpr(< 1,2 >' + < 3,4,5 >');
    ```

    would result in the vector $(4, 6, 6)$.

12. Note that the prime character (') is different from the single quotation mark character (‘) which can be used to enclose strings. The prime character can only be used with vectors, not with general matrices or matrix expressions. Note also that due to stack size limitations, a directly specified vector can have at most 9 elements. This may be less when the vector is used as part of a more complex expression.

13. In addition to its standard form, one can use the `mexpr` command in the form

    ```
    mexpr(<rsel,csel,a>, expression, Matrix-name)
    ```

    The optional string `<rsel,csel,a>` controls how the matrix expression is evaluated. `rsel` selects row indices, `csel` selects column indices of the matrix expression. Only elements selected both by `rsel` and `csel` are evaluated via expression, all other elements get a value defined by `a`.

    `a` can be a scalar constant or the name of an already existing matrix. `rsel` and `csel` can be specified as follows:

```
 *             selects all row (column) indices
 i             selects row (column) index i
-i             selects all rows (columns) except i
 (i1,i2,...)   selects rows (columns) i1,i2,...
-(i1,i2,...)   selects all rows (columns) expept i1,i2,...
 i(d)j         selects rows (columns) i, i+d, ...
-(i(d)j)       selects all rows (columns) except
               i, i+d, i+2d,...
```

Changing the Elements of a Matrix

In order to change specific elements of a matrix one can use the command

```
msetv(expression, X(row,col));
```

where `expression` is a scalar expression, `X` is the name of an existing matrix, and `row` and `col` are scalar expressions referring to an existing element of `X`. This element is then changed into the value of `expression`.

5.1.4 Matrix Commands

This section describes the currently available matrix commands.

5.1.4.1 Elementary Matrix Commands

This section describes some elementary matrix commands. All the commands create new matrices and will result in an error if there is not enough memory for the new matrices or if the maximum number of matrices is reached. The name of an existing matrix can be used to name a result. In this case the previously existing matrix is overwritten. Matrix commands do not echo unless `silent = -1`.

Number of Rows and Columns

1. Given a (m, n) matrix `A`, the command

   ```
   mnrow(A,R);
   ```

   creates a $(1, 1)$ matrix `R` that contains m.

2. Given a (m, n) matrix `A`, the command

   ```
   mncol(A,R);
   ```

   creates a $(1, 1)$ matrix `R` that contains n.

Multiplication, Transposition, Trace

1. The command

   ```
   mmul(A1,...,Ak,R);
   ```

   creates a matrix `R` that contains the product of the matrices `A1,...,Ak`.

2. The command

   ```
   mtransp(A,R);
   ```

   creates a matrix `R` that contains the transpose of the matrix `A`.

3. The command

   ```
   mtrace(A,R);
   ```

   creates a $(1, 1)$-matrix `R` that contains the trace of the matrix `A`.

4. The command

```
mcross(A,R);
```

creates a matrix `R` that contains the cross product `A'A`.

Diagonal Matrices

1. The command

```
mdiag(X,D);
```

assumes that `X` is a $(m, 1)$-matrix (column vector). The commands creates an (m, m)-matrix `D` and copies `X` into its main diagonal.

2. The command

```
mdrow(A,D);
```

assumes that `A` is a (m, n)-matrix. The commands creates an (m, m) diagonal matrix `D` and sets

$$d_{ii} = \sum_{j=1}^{n} a_{ij}$$

3. The command

```
mdcol(A,D);
```

assumes that `A` is a (m, n)-matrix. The commands creates an (n, n) diagonal matrix `D` and sets

$$d_{ii} = \sum_{i=1}^{m} a_{ij}$$

4. To extract the main diagonal from a matrix one can use the command

```
mdiagd(A,X);
```

It is assumed that `A` is a (m, n)-matrix. The commands creates a $(k, 1)$ column vector `X`, $k = \min(m, n)$, containing the main diagonal elements from `A`.

5. The command

   ```
   msqrtd(A,R);
   ```

   assumes that `A` is a (m, m) square matrix with non-negative entries in its main diagonal. The command creates a (m, m) diagonal matrix `R` and sets

   $$r_{ii} = \sqrt{a_{ii}}$$

6. The command

   ```
   msqrti(A,R);
   ```

   assumes that `A` is a (m, m) square matrix with non-negative entries in its main diagonal. The command creates a (m, m) diagonal matrix `R` and sets

   $$r_{ii} = 1/\sqrt{a_{ii}}$$

Concatenation

1. The command

   ```
   mcath(A1,...,Ak,R);
   ```

   creates a matrix

   $$\texttt{R} = [\texttt{A1}, \ldots, \texttt{Ak}]$$

   meaning that the matrices are horizontally concatenated. The number of rows of `R` will equal the number of rows of `A1`.

2. The command

   ```
   mcatv(A1,...,Ak,R);
   ```

   stacks the matrices `A1,...Ak` vertically into the new matrix `R`. The number of columns of `R` will equal the number of columns of `A1`.

3. The command

   ```
   mcathv(A1,...,Ak,R);
   ```

   creates the direct sum of the matrices `A1,...Ak`:

   $$\texttt{R} = \begin{bmatrix} \texttt{A1} & & \\ & \ddots & \\ & & \texttt{Ak} \end{bmatrix}$$

Sums of Rows and Columns

1. Given a (m, n)-matrix `A`, the command

   ```
   mrsum(A,R);
   ```

   creates a $(m, 1)$-matrix `R` that contains row sums of `A`.

2. Given a (m, n)-matrix `A`, the command

   ```
   mcsum(A,R);
   ```

   creates a $(1, n)$-matrix `R` that contains the column sums of `A`.

Note that row sums can also be computed with the command `mmul(A,1,R)`, and column sums with `mmul(1,A,R)`. Also, `mmul(1,A,1,R)` provides the sum of the elements of `A`.

Selection of Rows and Columns

1. The command

   ```
   msrow(A,D,R);
   ```

   assumes that `A` is a (m, n) matrix and `D` is a row or column vector providing indices for rows in `A`. Assuming that `D` contains k indices, the command creates (k, n)-matrix `R` that contains the selected rows from `A`.

2. The command

   ```
   mscol(A,D,R);
   ```

   assumes that `A` is a (m, n) matrix and `D` is a row or column vector providing indices for columns in `A`. Assuming that `D` contains k indices, the command creates (m, k)-matrix `R` that contains the selected columns from `A`.

Rows can also be specified by giving an index list in the form

`msrow(A,`$< i_1, i_2, \ldots >$`,R)`

Any sequence of integers between 1 and m can be specified. If elements in the list are separated by two consecutive commas the corresponding range is used. The same applies to `mscol`.

Trimming Matrices

The command

```
mtrim(A,ca,ra,cb,rb,R)
```

expects a (n, m) matrix `A` and scalar expressions `ca`, `ra`, `cb`, and `rb`. It creates a new matrix, `R`, by deleting the first `ca` columns of `A` (if `ca` ≥ 0) or adding `ca` zero columns (if `ca` < 0), and correspondingly treats the first `ra` rows of `A`, the last `cb` columns of `A`, and the last `rb` rows of `A`.

Stacking Rows or Columns of a Matrix

Given an (n, m) matrix `A`, the command

```
mcvec(A,V)
```

creates a $(n * m, 1)$ vector `V` containing the stacked columns of `A`. Correspondingly, the command

```
mrvec(A,V)
```

creates a $(n * m, 1)$ vector `V` containing the stacked rows of `A`.

Kronecker Product

Given matrices `A` and `B`, the command

```
mkp(A,B,R);
```

creates `R` $=$ `A` $\otimes$ `B`.

Permutations

The command

```
mprow(A,P,B)
```

expects an (n, m) matrix `A` $= (a_{ij})$ and an $(n, 1)$ or $(1, n)$ permutation vector `P` $= (p_i)$ and creates an (n, m) matrix `B` $= (b_{ij})$ with $b_{i,j} = a_{p_i,j}$. Analogously, the command

```
mpcol(A,P,B)
```

creates $b_{i,j} = a_{i,p_j}$. The command

```
mpsym(A,P,B)
```

expects a quadratic matrix `A` and creates the symmetrical permutation $b_{i,j} = a_{p_i,p_j}$. Finally, given an $(n,1)$ or $(1,n)$ permutation vector `P` $= (p_i)$, the command

```
mpinv(P,Q)
```

creates an output vector `Q` $= (q_i)$ with $q_{p_i} = i$, that is, the inverse permutation.

Matrix Aggregation

The command

```
mag(A,R,C,B)
```

expects an (n,m) matrix `A`, a $(n,1)$ vector `R` and a $(1,m)$ vector `C`. Elements of `R` correspond to the rows of `A`, elements of `C` correspond to columns of `A`. If these elements are zero the corresponding rows and/or columns of `A` are dropped. If two or more elements of `R` and `C` have the same value, the corresponding elements of `A` are aggregated. The rows and columns of the resulting matrix `B` are ordered according to the order of the elements of `R` and `C`, respectively. The following examples illustrate the command:

```
    1 2 3       1
A = 5 6 7   R = 0   C = 1 2 3   B = 2 4 4
    1 2 1       1

    1 2 3       1
A = 5 6 7   R = 0   C = 1 4 4   B = 2 8
    1 2 1       1

    1 2 3       1
A = 5 6 7   R = 0   C = 5 4 4   B = 8 2
    1 2 1       1
```

5.1.4.2 Sorting and Ranking

The following commands are available for sorting and ranking the rows of a matrix.

1. The command

   ```
   msort(A,D,R);
   ```

   assumes that `A` is a (m, n)-matrix and `D` is a column vector providing indices of columns of `A`. The command then sorts the rows of `A` in ascending order, based on the values in the columns selected by `D`, and returns the result in a new (m, n)-matrix `R`.

2. The command

   ```
   msort1(A,D,R);
   ```

   is almost identical with the `msort` command, but this command drops rows of `A` which occur more than once. Therefore, `R` will have the same number of columns but may have a smaller number of rows than `A`.

3. The command

   ```
   mrank(A,D,R);
   ```

   assumes that `A` is a (m, n)-matrix and `D` is a column vector providing indices of columns of `A`. The command then sorts the rows of `A` in ascending order, based on the values in the columns selected by `D`. The command finally returns a new $(m, 1)$-vector `R` that contains rank numbers for the sorted rows of `A`.

5.1.4.3 Standardization and Scaling

The following commands are available for centering, standardizing, and scaling matrices.

1. Given a (m, n)-matrix `A`, the command

   ```
   mcent(A,R);
   ```

   creates a (m, n)-matrix `R` and sets its columns to the mean-centered columns of `A`.

2. Given a (m, n)-matrix `A`, the command

   ```
   mstand(A,R);
   ```

   creates a (m, n)-matrix `R` and sets its columns to the standardized columns of `A`, that is, columns with mean 0 and variance 1. If a column has variance zero it is filled with zeros.

3. Given a symmetric (n, n)-matrix `A`, the command

   ```
   mdcent(A,R);
   ```

   creates a (n, n)-matrix `R` that contains the "double-centered" version of `A`, that is:

   $$r_{ij} = -\frac{1}{2}(a_{ij}^2 - a_{i.}^2 - a_{.j}^2 + a_{..}^2)$$

4. Given a (m, n)-matrix `A`, the command

   ```
   mscal1(A,R);
   ```

   creates a (m, n)-matrix `R` with values

   $$r_{ij} = a_{ij}/\Sigma_{kl} a_{kl}$$

5.1.4.4 Matrix Norms

1. Given a (m, n)-matrix `A`, the command

   ```
   mnorm(A,R);
   ```

   creates a $(1, 1)$-matrix `R` that contains the absolute value of that element of `A` that has largest absolute value.

2. Given a (m, n)-matrix `A`, the command

   ```
   mnorm1(A,R);
   ```

   creates a $(1, 1)$-matrix `R` that contains the sum of the absolute values of the elements of `A`.

3. Given a (m, n)-matrix `A`, the command

   ```
   mnorm2(A,R);
   ```

   creates a $(1, 1)$-matrix `R` that contains

   $$\sqrt{\Sigma_{ij} a_{ij}^2}$$

5.1.4.5 Inverse Matrices

Inversion of a Diagonal Matrix

Given a diagonal (m, n)-matrix `A`, the command

```
minvd(A,R);
```

returns a (m, n)-matrix `R` with non-zero elements

$$r_{ii} = 1/a_{ii} \qquad (i = 1, \ldots, \min(m, n))$$

Inversion of a Symmetric Positive Definite Matrix

Given a symmetric positive definite (n, n)-matrix `A`, the command

```
minvs(A,R);
```

returns a (n, n)-matrix `R` that contains the inverse of `A`. Values are taken only from the lower triangle of `A`, including its main diagonal.

Generalized Inverse Matrices

If A is a (m, n) matrix, there is a unique (n, m) matrix B with properties

$$ABA = A$$
$$BAB = B$$
$$(AB)' = AB$$
$$(BA)' = BA$$

B is then called the *generalized inverse* of A, or sometimes the *pseudo-inverse* or Moore-Penrose inverse of A. See, e.g., Pringle and Rayner [1971]. For more general definitions see Rao [1973, pp. 24–27]. If A is a square regular matrix, then its generalized inverse is identical with its standard inverse.

As shown by Golub and Reinsch [1971], calculation of the generalized inverse of A can be based on its singular value decomposition (see 5.1.4.12). If the singular value decomposition of A is given by

$$A = UQV'$$

Box 1 Example: generalized inverse

```
Command file: mat8.cf
mfmt = 13.10;
mdef(A,4,3) = 1,5,0, 2,6,0, 3,7,0, 4,8,0;
mpr(A);

mginv(A,B);
mpr(B);

mmul(A,B,C);
mmul(C,A,D);
mpr(D);

Output:
mdef(A,4,3)=...
mpr(A)
 1.0000000000  5.0000000000  0.0000000000
 2.0000000000  6.0000000000  0.0000000000
 3.0000000000  7.0000000000  0.0000000000
 4.0000000000  8.0000000000  0.0000000000
mginv(A,B)
Pseudorank of A: 2
mpr(B)
-0.5500000000 -0.2250000000  0.1000000000  0.4250000000
 0.2500000000  0.1250000000  0.0000000000 -0.1250000000
 0.0000000000  0.0000000000  0.0000000000  0.0000000000
mmul(A,B,C)
mmul(C,A,D)
mpr(D)
 1.0000000000  5.0000000000  0.0000000000
 2.0000000000  6.0000000000  0.0000000000
 3.0000000000  7.0000000000  0.0000000000
 4.0000000000  8.0000000000  0.0000000000
```

one can first define the generalized inverse of the diagonal matrix Q by

$$P_{ij} = \begin{cases} 1/Q_{ij} & \text{if } Q_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

In a second step, the generalized inverse of A can be calculated as

$$B = VPU'$$

TDA uses this method to calculate generalized inverse matrices. The singular value decomposition is done as described in **5.1.4.12**. The command is

```
mginv(A,B);
```

`A` is assumed to be an already defined (m, n) matrix with $m \geq n$. The procedure creates a (n, m) matrix `B` and the generalized inverse of `A` is returned in `B`. In addition, the rank of `A` is written in the standard output. Note that the numerical determination of the rank of a matrix always involves some inaccuracies due to limitations in representing floating point numbers on a digital computer. So in general, one only gets the pseudo-rank of a matrix.

Example 1 An illustration, using a $(3, 3)$ matrix, is given in Box 1. The result is shown in the same box.

5.1.4.6 Cholesky Decomposition

An (n, n) square matrix A is called *positive definite* if

$$x'Ax > 0 \qquad \text{for all } x \neq 0$$

If a matrix is symmetric and positive definite there is a unique decomposition

$$A = RR' \tag{1}$$

where R is a lower triangular matrix with positive diagonal elements. This is called a *Cholesky decomposition* of the matrix A, see, e.g., Mohamed and Wait [1986, p. 10], Martin et al. [1971].

TDA provides the command

```
mchol(A,R);
```

to calculate the Cholesky decomposition of `A`. This must be a (n, n) square matrix. TDA does not check whether the matrix is symmetric, it simply uses its lower triangular part to perform the decomposition. The result is then copied into the lower triangular part of the (n, n) matrix `R`. If `A` is not positive definite, there will be an error message and `R` = 0.

5.1.4.7 Least Squares Regression

In matrix notation, a linear regression model can be written as

$$Y = X\beta + \epsilon \tag{1}$$

with an $(m, 1)$ vector Y (dependent variable), an (m, n) matrix X (independent variables), an $(n, 1)$ vector β (regression coefficients to be estimated), and an $(m, 1)$ vector ϵ (residuals). A least squares solution of (1) is a vector $\hat{\beta}$ that minimizes the Euclidean norm of the residuals:

$$\| \epsilon \| = \| Y - X\beta \| \tag{2}$$

Here we use $\| x \|$ to denote the Euclidean norm of a vector x, that is, the square root of Σx_i^2. In the context of linear regression, it is normally assumed that $m > n$ and X has full column rank ($\text{rank}(X) = n$). There is, then, a unique least squares solution

$$\hat{\beta} = (X'X)^{-1}X'Y \tag{3}$$

If $\text{rank}(X) < n$, there are (in general) many possible solutions. To calculate a least squares solution for (1), TDA uses an algorithm developed by Hanson and Haskell [1982]. (This algorithm extends earlier work of Lawson and Hanson [1974].) One should note that this algorithm does not require that X has full column rank. If $\text{rank}(X) < n$, the algorithm tries to find one of the possible solutions. The command is

```
mls(S,B);
```

with `S` an $(m, n + 1)$ matrix containing $[X, Y]$. If the input data are not already available in such a matrix, `S` can easily be created by using the `mcath` command; see 5.1.4.1. The contents of `S` is destroyed by the command. The command creates an $(n, 1)$ vector `B` for the least squares solution, β, of (1). In addition, the rank of X and the Euclidean norm of the residuals (2) are shown in the standard output.

Box 1 Example of least squares regression

```
Command file: mat2.cf
mfmt = -16.12;
mdef(A,4,3) = 1,2,3, 4,7,11, -1,1,0, 5,6,11;
mpr(A);
mls(A,B);
mpr(B);

Output:
mdef(A,4,3)=...
mpr(A)
  1.000000000000e+00   2.000000000000e+00   3.000000000000e+00
  4.000000000000e+00   7.000000000000e+00   1.100000000000e+01
 -1.000000000000e+00   1.000000000000e+00   0.000000000000e+00
  5.000000000000e+00   6.000000000000e+00   1.100000000000e+01
mls(A,B)
Rank of left-hand side: 2
Norm of residuals:   4.440892098501e-16
mpr(B)
  1.000000000000e+00
  1.000000000000e+00
```

Example 1 Box 1 illustrates the `mls` command with a simple example. The command file is `mat2.cf`.

5.1.4.8 Linear Equations

A general system of linear equations can be written as

$$Ax = b \tag{1}$$

where A is an (m, n) left-hand side matrix, b is an $(m, 1)$ right-hand side matrix (vector), and x is an $(n, 1)$ matrix (vector) to be determined as the solution of the system. The command

```
mlse(S,X);
```

can be used to find a solution of the system (1). `S` should be a $(m, n+1)$ matrix containing $[A, b]$. The command creates an $(n, 1)$ vector `X` and, if a solution is found, it is copied into this matrix. In addition, one gets information about the rank of the left-hand side matrix, A, and the Euclidean norm of the residuals:

$$\| Ax - b \|$$

To solve system (1), TDA uses an algorithm developed by Hanson and Haskell [1982]. The algorithm is quite general and can cope also with under-determined and over-determined systems. In general, there are three possible situations.

1. A is a square matrix ($m = n$) and has full rank. Then there is a unique solution x that solves the system (1). An example (taken from Lawson and Hanson [1974, p. 14]) is shown in Box 1.

2. If $m \leq n$ and $\text{rank}(A) < n$, there may be a unique solution, or a set of different solutions, or the system may be contradictory without a solution. If there is more than one solution, TDA will provide just one of the possible solutions. If the equations are contradictory, TDA calculates a least squares solution.

3. If $m > n$ and $\text{rank}(A) \geq n$, the algorithm tries to find a least squares solution of the system, that is, a vector x that minimizes the Euclidean norm of the residuals.

Box 1 Example: full rank system of linear equations

```
Command file: mat3.cf  (Lawson, Hanson 1974, p. 14)
mfmt = 12.8;
mdef(A,3,4) = -0.4744,-0.4993,-0.7250,-0.8615,
               0.5840,-0.7947, 0.1652, 0.0,
              -0.6587,-0.3450, 0.6687, 0.0;
mpr(A);
mlse(A,B);
mpr(B);

Output:
mdef(A,3,4)=...
mpr(A)
 -0.47440000  -0.49930000  -0.72500000  -0.86150000
  0.58400000  -0.79470000   0.16520000   0.00000000
 -0.65870000  -0.34500000   0.66870000   0.00000000
mlse(A,B)
Rank of left-hand side: 3
Norm of residuals:   0.00000000
mpr(B)
  0.40872542
  0.43019126
  0.62456022
```

5.1.4.9 Linear Inequalities

A general system of linear inequalities can be written as

$$Ax \geq b \tag{1}$$

where A is an (m, n) left-hand side matrix, b is an $(m, 1)$ right-hand side matrix (vector), and x is an $(n, 1)$ matrix (vector) to be determined as the solution of the system. In general, this system of inequalities may have a unique solution, many solutions, or no solution if the inequalities are incompatible. To solve the system, TDA uses again an algorithm developed by Hanson and Haskell [1982]. This algorithm always tries to find (at least) one solution with a minimal Euclidean norm of the residuals. The command is

```
mlsi(S,X);
```

with `S` an $(m, n + 1)$ matrix containing $[A, b]$. The command creates an $(n, 1)$ matrix `X` and, if a solution is found, it is copied into this matrix. The contents of the matrix `S` is destroyed by the procedure. Box 1 illustrates the `mlsi` command with a simple example.

Box 1 Example: linear inequalities

```
Command file: mat4.cf
mfmt = 12.8;
mdef(A,4,3) = 1,2,3,5,6,11,-1,-2,-3,0,1,1;
mpr(A);
mlsi(A,B);
mpr(B);

Output:
mdef(A,4,3)=...
mpr(A)
  1.00000000   2.00000000   3.00000000
  5.00000000   6.00000000  11.00000000
 -1.00000000  -2.00000000  -3.00000000
  0.00000000   1.00000000   1.00000000
mlsi(A,B)
mpr(B)
  1.00000000
  1.00000000
```

5.1.4.10 Regression with Constraints

As described in 6.9.1, the `lsreg` command can solve least squares problems with additional equality and inequality constraints. To allow for these calculations with the help of matrix commands, we added the following commands.

1. The command

   ```
   mlsei(W,me,mi,X);
   ```

   expects a $(m, n+1)$ matrix `W` and two scalar expressions, `me` and `mi`, in the following form:

   $$\mathtt{W} = \begin{pmatrix} E & F \\ A & B \\ H & G \end{pmatrix}$$

   where

   E is a $(\mathtt{me}, n)$ matrix

   F is a $(\mathtt{me}, 1)$ vector

   A is a $(\mathtt{ma}, n)$ matrix

   B is a $(\mathtt{ma}, 1)$ vector

   H is a $(\mathtt{mi}, n)$ matrix

   G is a $(\mathtt{mi}, 1)$ vector

   and $\mathtt{ma} = m - \mathtt{me} - \mathtt{mi}$. Note that m is defined by the number of rows in `W`, and `me` and `mi` are given by the scalar expressions in the command. It is possible to have $\mathtt{me} \geq 0$, $\mathtt{ma} \geq 0$, and $\mathtt{mi} \geq 0$. The command tries to find (not always successfully) an $(n, 1)$ vector `X` in such a way that

   $$EX = F \qquad \text{(equality constraints)}$$

   $$HX \geq G \qquad \text{(inequality constraints)}$$

   and the Euclidean norm of $(AX - B)$ is minimal in the least squares sense. The algorithm is the same as used for TDA's `lsreg` command.

2. The command

```
mnls(W,me,k,X);
```

expects a $(m, n+1)$ matrix `W` and two scalar expressions, `me` and `k`, in the following form:

$$\texttt{W} = \begin{pmatrix} E & F \\ A & B \end{pmatrix}$$

where

E is a $(\texttt{me}, n)$ matrix

F is a $(\texttt{me}, 1)$ vector

A is a $(\texttt{ma}, n)$ matrix

B is a $(\texttt{ma}, 1)$ vector

and $\texttt{ma} = m - \texttt{me}$. Note that m is defined by the number of rows in `W`, and $\texttt{me} \geq 0$ is given by the scalar expressions in the command. The command tries to find (not always successfully) an $(n, 1)$ vector `X` in such a way that

$$EX = F \qquad \text{(equality constraints)}$$

and the Euclidean norm of $(AX - B)$ is minimal in the least squares sense. In addition, k is a scalar expression that satisfies $0 \leq k \leq n$ and is used to specify non-negativity constraints for the solution vector, X, in such a way that

$$X(j) \geq 0 \quad \text{for} \;\; j = k, \ldots, n$$

The algorithm is the same as used for TDA's `lsreg` command.

5.1.4.11 Eigenvalues and Eigenvectors

Let A be a square (n,n) matrix. An $(n,1)$ vector $x \neq 0$ is called an *eigenvector* of A, if there is a scalar λ so that

$$Ax = \lambda x \tag{1}$$

λ is then called an *eigenvalue* (or *latent root*) of A, and x is the corresponding *eigenvector* (or *latent vector*). These concepts play a fundamental role in linear algebra and also have statistical applications. Most linear algebra textbooks, and many statistical textbooks, introduce the basic concepts and theorems. Here we only note a few points.

1. Eigenvectors defined by (5.1.4.11) are *right* eigenvectors. Correspondingly, one can define *left* eigenvectors as non-trivial solutions of

   $$yA = \lambda y$$

   Left eigenvectors have dimension $(1,m)$ (row vectors). By transposing (1) one sees that the transposed right eigenvectors of A are the left eigenvectors of A'.

2. For every (n,n) matrix A one can define a *characteristic equation*

   $$\mid A - \lambda I_n \mid = 0 \tag{2}$$

   where $\mid\ \mid$ denotes the determinant of a matrix. The eigenvalues of A are the roots of this characteristic equation.

3. Every nonzero (n,n) matrix possesses at least one nonzero eigenvector and a corresponding eigenvalue. Or put otherwise, one can say that every (n,n) matrix has n eigenvalues; but these eigenvalues are not necessarily distinct, there can be multiple eigenvalues (roots of the characteristic equation). For example, let A be the $(2,2)$ matrix

   $$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \tag{3}$$

   The characteristic equation is

   $$\mid A - \lambda I_2 \mid = \begin{vmatrix} 1-\lambda & 1 \\ 0 & 1-\lambda \end{vmatrix} = (1-\lambda)^2$$

   Obviously, there is only a single root, 1.

4. A general (n, n) matrix A can have real or complex eigenvalues and eigenvectors. Complex eigenvalues and eigenvectors always occur in conjugate complex pairs.[1] If $\lambda = \lambda^{(r)} + i\lambda^{(i)}$ is a complex eigenvalue of A, and $x = x^{(r)} + ix^{(i)}$ the associated complex eigenvector, then

$$\begin{aligned} A(x^{(r)} + ix^{(i)}) &= (\lambda^{(r)} + i\lambda^{(i)})(x^{(r)} + ix^{(i)}) \\ &= (\lambda^{(r)}x^{(r)} - \lambda^{(i)}x^{(i)}) + i(\lambda^{(r)}x^{(i)} + \lambda^{(i)}x^{(r)}) \end{aligned}$$

It follows that

$$\begin{aligned} \lambda^* x^* &= (\lambda^{(r)}x^{(r)} - \lambda^{(i)}x^{(i)}) - i(\lambda^{(r)}x^{(i)} + \lambda^{(i)}x^{(r)}) \\ &= Ax^{(r)} - iAx^{(i)} \\ &= Ax^* \end{aligned}$$

That is, if λ is a complex eigenvalue of A with complex eigenvector x, then also λ^* is an eigenvalue of A with associated eigenvector x^*.

5. Eigenvectors are not unique. As seen from (1), if x is an eigenvector of A, then also αx is an eigenvector where α is any nonzero scalar. Therefore, when calculating eigenvalues, one has to adopt a convention for scaling. The TDA algorithm (described below) uses the following convention: If an eigenvector is real, it is scaled to have the Euclidean length 1, that is

$$\| x \| = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2} = 1 \tag{4}$$

If an eigenvector is complex, say $x = x^{(r)} \pm ix^{(i)}$, it is scaled so that

$$\max_{1 \le i \le n} \left\{ x_i^{(r)} \right\} = 1$$

and the imaginary component, corresponding to the maximum real part component, gets the value zero.

6. If the (n, n) matrix A is symmetric ($A = A'$), then all of its eigenvalues and eigenvectors are real. If, in addition, the matrix is positive definite (semi-definite), all eigenvalues are positive (not negative).

[1] If $z = a + ib$ is a complex number, then the complex conjugate of z, denoted z^*, is defined as $z^* = a - ib$.

Box 1 Example: eigenvectors and eigenvalues

```
Command file: mat5.cf

mfmt = 13.10;
mdef(A,4,4) = 6,-3,4,1, 4,2,4,0, 4,-2,3,1, 4,2,3,1;
mpr(A);
mev(A,ER,EI,EVR,EVI);
mpr(ER);
mpr(EI);
mpr(EVR);
mpr(EVI);

Output:

mdef(A,4,4)=6,-3,4,1,4,2,4,0,4,-2,3,1,4,2,3,1
mpr(A)
 6.0000000000 -3.0000000000  4.0000000000  1.0000000000
 4.0000000000  2.0000000000  4.0000000000  0.0000000000
 4.0000000000 -2.0000000000  3.0000000000  1.0000000000
 4.0000000000  2.0000000000  3.0000000000  1.0000000000
mev(A,ER,EI,EVR,EVI)
mpr(ER)
 5.2360679292
 5.2360680258
 0.7639320225
 0.7639320225
mpr(EI)
 0.0000000000
 0.0000000000
 0.0000000207
-0.0000000207
mpr(EVR)
-0.2627643879 -0.2627643970 -0.3726779962 -0.3726779962
-0.6152998190 -0.6152998181  0.1273220038  0.1273220038
-0.2350236149 -0.2350236199  0.3333333333  0.3333333333
-0.7050708544 -0.7050708501  1.0000000000  1.0000000000
mpr(EVI)
 0.0000000000  0.0000000000  0.0000000122 -0.0000000122
 0.0000000000  0.0000000000  0.0000000060 -0.0000000060
 0.0000000000  0.0000000000 -0.0000000134  0.0000000134
 0.0000000000  0.0000000000 -0.0000000000  0.0000000000
```

General Matrices

TDA offers two algorithms to calculate eigenvalues and eigenvectors. One algorithm is restricted to symmetric matrices and will be described be-

Box 2 Example: eigenvalues of symmetric matrix

```
Command file: mat6.cf

silent=-1;
mfmt = 13.10;
mdef(A,3,3) = 1,1,1, 1,2,0, 1,0,10;
mpr(A);

mevs(A,E,V);                calculate eigenvalues and vectors
mpr(E);                     print eigenvalues
mpr(V);                     print eigenvectors

mmul(A,V,B);                check results
mpr(B);
mscol(V,1,V1);
mexpr(E(1,1)*V1,B1);
mpr(B1);
mscol(V,2,V2);
mexpr(E(2,1)*V2,B2);
mpr(B2);
mscol(V,3,V3);
mexpr(E(3,1)*V3,B3);
mpr(B3);
```

low. The other algorithm is adopted from Grad and Brebner [1963] and can be applied to arbitrary real quadratic matrices. The command is

```
mev(A,ER,EI,EVR,EVI);
```

The command expects a (n, n) matrix `A` and creates four matrices `ER`, `EI`, `EVR`, and `EVI`. `ER` and `EI` are both $(n, 1)$ vectors containing the real and imaginary parts of the eigenvalues, respectively. `EVR` and `EVI` are (n, n) matrices and will contain the real and imaginary parts of the right eigenvectors, respectively. Note that the eigenvectors are scaled as described above.

Example 1 To illustrate the `mev` command, we use an example given in Eberlein and Boothroyd [1971, p. 334]. The data and results are shown in Box 1. The matrix A is defective and has two pairs of multiple roots: $3 \pm \sqrt{5}$. The right eigenvectors are:

$$(\pm\sqrt{5}, 3 \pm \sqrt{5}, 2, 6\,)'$$

The output of the `mev` command in Box 1 shows the scaled eigenvectors. As a result of rounding errors, the imaginary parts of the second pair

of eigenvalues and eigenvectors are not exactly zero. These eigenvectors are therefore scaled according to the complex case mentioned above.

Symmetric Matrices

A quadratic (n, n) matrix A is called *symmetric* if $A = A'$, that is, if A is equal to its transpose. As already mentioned, symmetric matrices only have real eigenvalues and eigenvectors and, therefore, the calculation is easier than for general matrices. To calculate eigenvalues and eigenvectors of a symmetric matrix, TDA offers the command

```
mevs(A,E,V);
```

It is expected that `A` is an (n, n) symmetric matrix. The procedure creates an $(n, 1)$ matrix (vector) `E` and an (n, n) matrix `V`. The eigenvalues of `A` are stored in `E` (in descending order). The eigenvectors of `A` are stored in the columns of `V`. The eigenvector corresponding to the ith eigenvalue (ith component of `E`) is stored in the i column of `V`. Eigenvectors are scaled to have the Euclidean length 1.

The algorithm to calculate eigenvalues and eigenvectors is adopted from Sparks and Todd [1974]. Note that the algorithm does not check whether the input matrix, `A`, is symmetric; only its lower triangular part is used.

Example 2 To illustrate the `mevs` command, Box 2 shows a simple example, supplemented with commands to check the result. The output is shown in Box 3.

Box 3 Example (continued): eigenvalues of symmetric matrix

```
Output of mat6.cf

mdef(A,3,3)=1,1,1,1,2,0,1,0,10
mpr(A)
 1.0000000000  1.0000000000  1.0000000000
 1.0000000000  2.0000000000  0.0000000000
 1.0000000000  0.0000000000 10.0000000000
mevs(A,E,V)
mpr(E)
10.1112597744
 2.5823542359
 0.3063859897
mpr(V)
 0.1105672008 -0.5020854553 -0.8577208693
 0.0136313229 -0.8621650266  0.5064441272
 0.9937751663  0.0676879796  0.0884830847
mmul(A,V,B)
mpr(B)
 1.1179736900 -1.2965625022 -0.2627936574
 0.1378298465 -2.2264155084  0.1551673851
10.0483188637  0.1747943409  0.0271099775
mscol(V,1,V1)
mexpr(E(1,1)*V1,B1)
mpr(B1)
 1.1179736900
 0.1378298465
10.0483188637
mscol(V,2,V2)
mexpr(E(2,1)*V2,B2)
mpr(B2)
-1.2965625022
-2.2264155084
 0.1747943409
mscol(V,3,V3)
mexpr(E(3,1)*V3,B3)
mpr(B3)
-0.2627936574
 0.1551673851
 0.0271099775
```

5.1.4.12 Singular Value Decomposition

If A is an (m, n) matrix with $m \geq n$, there is a decomposition

$$A = UQV' \tag{1}$$

with the following matrices (I_n is used to denote the (n, n) unit matrix): an orthogonal (m, n) matrix U, i.e.,

$$U'U = I_n$$

An orthogonal (n, n) matrix V, i.e.,

$$V'V = VV' = I_n$$

And a (n, n) diagonal matrix

$$Q = \text{diag}\{q_1, \ldots, q_n\}$$

(1) is called the singular value decomposition of the matrix A; and the entries $q_1, \ldots, q_n$ are called the singular values of A. See, e.g., Golub and Reinsch [1971]. Basic properties are:

1. The number of nonzero singular values equals the rank of A.
2. $q_1^2, \ldots, q_n^2$ are the eigenvalues of the matrix $A'A$.
3. The columns of V are the eigenvectors of $A'A$, that is:

$$(A'A)V = (QQ)V$$

The singular value decomposition is a basic tool for linear algebra problems but also has many statistical applications.

Box 1 Example: singular value decomposition

```
Command file: mat7.cf
mfmt = 13.10;
mdef(A,3,3) = 1,0,0, 5,7,0, 1,0,1;
mpr(A);

mcross(A,AA);        create cross product
mpr(AA);

msvd1(AA,Q,U,V);     singular value decomposition
mpr(Q);
mpr(U);
mpr(V);

mevs(AA,E,V);        calculate eigenvalues and eigenvectors
mpr(E);
mpr(V);
```

TDA provides two commands for singular value decomposition. Both assume a (m, n) matrix, `A`, with $m \geq n$. The command

```
msvd(A,Q);
```

performs a singular value decomposition of `A` and saves the singular values in the $(n, 1)$ matrix `Q`. The command

```
msvd1(A,Q,U,V);
```

does the same, but additionally creates the (m, n) matrix `U` and the (n, n) matrix `V` as described above. Calculation is based on an algorithm by Golub and Reinsch [1971].

Example 1 To illustrate the `msvd1` command, Box 1 shows a simple example. To check the result, the `mevs` command is used to calculate eigenvalues and eigenvectors directly. The output is shown in Box 2.

Box 2 Example (continued): singular value decomposition

```
Output of mat7.cf
mdef(A,3,3)=...
mpr(A)
 1.0000000000  0.0000000000  0.0000000000
 5.0000000000  7.0000000000  0.0000000000
 1.0000000000  0.0000000000  1.0000000000
mcross(A,AA)
mpr(AA)
27.0000000000 35.0000000000  1.0000000000
35.0000000000 49.0000000000  0.0000000000
 1.0000000000  0.0000000000  1.0000000000
msvd1(AA,Q,U,V)
mpr(Q)
74.6926238609
 1.9752562176
 0.3321199215
mpr(U)
-0.5917332901  0.6195362372  0.5157776305
-0.8060938137 -0.4611140127 -0.3709267188
-0.0080297492  0.6352548449 -0.7722608401
mpr(V)
-0.5917332901  0.6195362372  0.5157776305
-0.8060938137 -0.4611140127 -0.3709267188
-0.0080297492  0.6352548449 -0.7722608401
mevs(AA,E,V)
mpr(E)
 0.3321199215
 1.9752562176
74.6926238609
mpr(V)
-0.5157776305 -0.6195362372 -0.5917332901
 0.3709267188  0.4611140127 -0.8060938137
 0.7722608401 -0.6352548449 -0.0080297492
```

5.1.4.13 Linear Programming

A linear programming problem can be described as follows:

$$\begin{array}{rcl} c'x & = & z \rightarrow \max \\ Ax & \leq & b \\ x & \geq & 0 \end{array} \tag{1}$$

The first line defines the objective function to be maximized; c is an $(n, 1)$ vector with given values, x is an $(n, 1)$ vector containing variables. Explicitly written, the objective function is

$$z(x_1, \ldots, x_n) = \sum_{i=1}^{n} c_i x_i$$

The second and third lines in (1) define constraints. The second line assumes a given (m, n) matrix A and an $(m, 1)$ vector b resulting in m inequalities to be satisfied by x. The third line requires that feasible solutions for x are restricted to be nonnegative.

(1) is called the *primal* problem. Associated with the primal problem is a *dual problem* defined as

$$\begin{array}{rcl} b'y & = & z \rightarrow \min \\ A'y & \geq & c \\ y & \geq & 0 \end{array} \tag{2}$$

The variables are now given in an $(m, 1)$ vector y. The objective function, $b'y$, is given the same value, z, because it can be shown that, if the primal problem has a (finite) solution, then also the dual problem has a solution, and the values of the primal and dual objective functions are the same.

The formulation in (1) assumes that there is only a specific type of inequality constraints: $Ax \leq b$. However, if in an original problem formulation, some constraints are formulated with an $\geq$ relation, that equation can simply be multiplied by -1 to fit into the formulation given in (1). It would also be possible to use two inequality constraints to define an equality to hold exactly, but equality constraints can also defined directly.

The algorithm is adopted from Salazar and Sen [1968]. The TDA command can be given in two different forms. A first command is

```
mlp(T,X,Y);
```

`T` is expected to be a $(m+1, n+1)$ matrix containing the data for the linear programming problem in the following form:

$$T = \begin{bmatrix} c' & 0 \\ A & b \end{bmatrix}$$

with A, b, and c as defined above. If possible, an optimal solution of the primal and dual problems is calculated. Alternatively, the command can be given as

```
mlp1(T,p,X,Y);
```

The last p rows of $Ax \leq b$ are then interpreted as equality constraints.

Box 1 Example: linear programming

```
Command file: lp1.cf
mdef(T,3,3) =  1, 0, 0,
               1, 0, 1,
               0, 1, 1;
mpr(T);
mlp(T,X,Y,1);
mpr(X);
mpr(Y);

Output:
mdef(T,3,3)=...

mpr(T)
      1.0000       0.0000       0.0000
      1.0000       0.0000       1.0000
      0.0000       1.0000       1.0000

mlp(T,X,Y,1)
Value is:       1.0000

mpr(X)
      1.0000
      1.0000

mpr(Y)
      1.0000
      0.0000
```

Example 1 A trivial example is shown in Box 1. The problem is to maximize x_1, s.t. $x_1, x_2 \geq 0$, $x_1 \leq 1$, and $x_2 = 1$.

5.1.4.14 Some Further Commands

1. Given $(m, 1)$-vectors `T` and `C`, the command

   ```
   mple(T,C,F,D);
   ```

   performs a product-limit estimation of the cumulative distribution function of the values in `T`. The elements in `C` are interpreted as censoring information: if `C(i) = 0`, `T(i)` is interpreted as *not* censored; if `C(i)` has a non-zero value, the corresponding `T(i)` is interpreted as censored. The largest value of `T` is always interpreted as being *not* censored.

   The command creates two $(m, 1)$-vectors, `F` and `D`. If the command terminated successfully, `F` will contain the cumulative distribution function, and `D` will contain the jumps of the function for uncensored values.

2. Given $(m, 1)$-vectors `A` and `W`, the command

   ```
   mwvec(A,W,R);
   ```

   creates a $(m, 1)$-vector `R` with elements

   $$r_i = \frac{\sum_{j \in I(i)} a_j w_j}{\sum_{j \in I(i)} w_j} \quad \text{where} \quad I(i) = \{k \,|\, i < k \leq m\}$$

   If $I(i)$ is empty, or the sum of weights is zero, then $r_i = a_i$.

3. Given $(m, 1)$-vectors `A`, `W`, and `T`, the command

   ```
   mwvec1(A,W,T,R);
   ```

   creates a $(m, 1)$-vector `R` with elements

   $$r_i = \frac{\sum_{j \in I(i)} a_j w_j}{\sum_{j \in I(i)} w_j} \quad \text{where} \quad I(i) = \{k \,|\, T(k) > T(i)\}$$

   If $I(i)$ is empty, or the sum of weights is zero, then $r_i = a_i$.

4. Given a (n, n) matrix `A`, the command

   ```
   mpz(A,B,P);
   ```

   tries to find a permutation of the rows of `A` that puts as many nonzero elements in the main diagonal as possible. The permuted matrix is returned in `B`. Also returned is an $(n, 1)$ vector `P` that contains the permutation. (The algorithm is adapted from ACM algorithm 575 (Permutations for a zero-free diagonal), developed by I.S. Duff.)

5. Given a (n, n) matrix `A`, the commands

   ```
   mpbl(A,B,P,N,U)
   mpbu(A,B,P,N,U)
   ```

   try to find a simultaneous permutation of rows and columns to make `A` a lower (`mpbl`) or upper (`mpbu`) block diagonal matrix. The permuted matrix is returned in `B`. Also returned is an $(n, 1)$ vector `P` that contains the permutation, an $(1, 1)$ scalar `N` that contains the number of blocks, and a $(k, 1)$ vector `U` with k the number of blocks. $u_{i,1}$ is the number of the row where the ith block begins. (The algorithm is adapted from ACM algorithm 529 (Permutations to block triangular form), developed by I.S. Duff and J.K. Reid.)

6. The command

   ```
   mpit(F,N,k,R)
   ```

   can be used to iterate a Leslie matrix. It is assumed that `F` is a $(n, 2)$ matrix, `N` is a $(n, 1)$ vector, and k is an integer. The first column of `F` contains the first row of the (n, n) Leslie matrix L, and the second column of `F` contains the subdiagonal of L (the last element is ignored). The vector `N` contains the starting values for the iteration. The command creates the $(k + 1, n)$ matrix `R`. The ith row of `R` contains L^i `N`.

7. The command `mproc` (Procrustes rotation) is described in Section 7.4.4.1.

8. The parameters `mplog`, `mppar`, `mpcov`, and `mpgrad` (giving results for function maximization) are described in Section 5.6.1.

9. The following commands are only documented in the help file `tda.hlp`:
   a) `mbrr` (results of balanced replications)
   b) `mcel` (edge list from adjacency matrix)
   c) `mpfit` (iterated proportional fitting)
   d) `mqap` (quadratic assignment)

5.2 Expressions

This chapter explains TDA's concept of expressions. It contains the following sections.

5.2.1 Introduction

Many TDA commands require *expressions*, that is, strings which can be evaluated numerically; for example:

`1.2`	a single floating point number
`max(1,7)`	maximum of two numbers
`exp(expression)`	exponential function of another expression

In TDA, expressions serve mainly three purposes:

1. To define variables. In general, a variable is defined by

   ```
   VName = expression;
   ```

   `VName` is the name of the variable (to be defined), and `expression` describes how to assign numerical values to the variable (for additional options see 2.1). Remember that a variable is actually a column vector with a number of components equal to the number of rows in the data matrix. Consequently, `expression` is evaluated for each row in the data matrix and the resulting numerical value is the value of the variable for that row.

2. To define functions. This will be explained in 5.3.

3. To specify case select statements. For instance, the syntax of the `isel` (input select) command is

   ```
   isel = expression;
   ```

   meaning that `expression` is evaluated for each record of the input data file. The record is then selected (for the data matrix) if the result is not equal to zero, and otherwise the record is not selected.

In general, expressions are build by using numerical constants and random numbers, references to data file entries or already defined variables, and by using operators. Several operators will be introduced in subsequent sections. There are operators without any arguments and operators that have one or more arguments. For instance, `rd` is an operator without arguments and provides the next random number, equally distributed in

the 0–1 interval, from the connected random number generator. An example of an operator with two arguments is `rd(a,b)`; the arguments `a` and `b` are real numbers with `a < b`. This operator provides the next random number that is equally distributed in the interval [a,b].

In general, most operators are identified by a key word, for instance `rd`, that *must be given in lower case letters*. If an operator has one or more arguments, these arguments follow in brackets, separated by commas:

```
op(arg1,arg2,...)
```

Some operators have a somewhat different syntax. For instance, the simple arithmetical operators (+, -, ...) are operators with two arguments. They are, however, not written as `+(x,y)`, but more conveniently as `x + y`. Another exception are the operators to create dummy variables.

Most operators have none or a fixed number of arguments. Some operators have a variable number of arguments, in particular:

`min`$(a_1, a_2, \dots)$ minimum of $a_1, a_2, \dots$
`max`$(a_1, a_2, \dots)$ maximum of $a_1, a_2, \dots$

An important distinction is based on the amount of information required for the evaluation of operators. In TDA, the basic distinction is between type 1 and type 2 operators.

1. Type 1 operators can be evaluated using only information from the current data file record or data matrix row. Therefore, these operators can be evaluated while sequentially reading an input data file and creating the rows of the data matrix.

2. Type 2 operators are operators that require information about the values of their arguments for all cases of the data matrix (if in *record mode*) or for all records in a block (if in *block mode*). For instance, `mean(V)`, the mean of a variable `V`, requires all values of `V` in the current data matrix (or data block) for its evaluation. There are, therefore, some restrictions in using these operators, see 5.2.6.

5.2.2 Evaluating Expressions

Expressions are evaluated in two steps. In a first parsing step, TDA creates an algorithm to evaluate the expression. This is actually a stack that symbolically references all operators and arguments contained in the expression. In a second step the algorithmic representation of the expression can be evaluated for any number of different arguments.

1. Understanding details of the first step is actually not necessary when using expressions. However, TDA offers the command

```
parse = expression;
```

to get some information about the result of the parsing procedure.

2. A more interesting command is

```
mpr(expression);
```

This command evaluated `expression` and prints the result into the standard output. Optionally, the command can also be used as

```
mpr(expression,string);
```

where `string` is an arbitrary string. The command then first prints the string, in a new line, prints the value of expression. As a further option, the command can be used as

```
mpr(expression [,string]) = file_name;
```

The result is then written, not to the standard output, but to the file specified on the right-hand side. By default, a new file is created by the `mpr` command. Using the command as `mpra`, data can be appended to an already existing file.

5.2.3 Numerical Constants

Numerical constants can be used to create expressions. They can be given in the following formats:

Integer	for example: 0, 367, -133
Floating point F Format	for example: 11.87, -1.1, -.12
Floating point E Format	for example: 1.3E12, -1E-123, 1.e+2

Numerical constants must not begin with a + sign. For instance the expression +3 will result in a syntax error because + is interpreted as an operator. One should also note that, if numerical constants are used to define variables, the possible accuracy depends on the *storage size* of the variable (see 2.1). However, if a variable is defined as a single numerical constant, the storage size is always 8 for a double precision value.

Some numerical constants have predefined names.

1. The operator `pi` (without arguments) provides the numerical constant $\pi = 3.14159\ldots$

5.2.4 Random Numbers

The following operators can be used to generate random numbers.

1. The operator `rd` provides the next random number, equally distributed in the 0–1 interval. The algorithm is adopted from Pike and Hill [1965], modified for 32 bit machines.

2. The operator `rd(a,b)` provides the next random number, equally distributed in the $[a, b]$ interval. The algorithm is the same as used for the `rd` operator.

3. The operator `grd(V)` provides a random number that is equally distributed in the 0–1 interval; a new random number is only generated in the first row of the data matrix or if the variable `V` changes its value. The algorithm is the same as used for the `rd` operator. Note that this operator is classified as a type 1 operator but has some restrictions. In particular, this operator should not directly be used as part of an if-then-else construction.

4. The operator `rdn` provides standard normally distributed random numbers. The algorithm is adopted from Bell [1968].

5. The operator `rdn1` provides an alternative method for standard normally distributed random numbers, based on an algorithm described by Brent [1974].

6. The operator `rdmn`(i, Σ) creates random numbers that follow a multivariate normal distribution with correlation matrix Σ. The operator requires that a complete symmetric and positive definite correlation matrix is defined with the `mdef` command, see 5.1.2. Given that an (n, n) correlation matrix Σ has been defined, `rdmn`(i, Σ) returns the ith component of

   $$(\epsilon_1, \ldots, \epsilon_n) \sim \mathcal{N}(0, \Sigma)$$

   It is required that $n \geq 2$. A new random vector is only created when $i = 1$, otherwise the operators returns components of a previously generated random vector.

   The algorithm follows Tong [1990, p. 185]. Its input consists of standard normally distributed random numbers that are created with the

algorithm of Brent [1974].

7. The operator `rdp1(z)` returns random numbers according to a Poisson distribution with mean `z`. The range is $0 < z < 350$. The algorithm is adapted from Schaffer [1970].

5.2.5 Type 1 Operators

Type 1 operators can be evaluated by using only information from the current data file record or data matrix row. The available type 1 operators are described in the following subsections.

5.2.5.1 General Type 1 Operators

1. The operator `case` returns the current case number, that is, the sequence number of the data matrix row. For example,

   ```
   CASE [6.0] = case;
   ```

   defines a variable, `CASE`, with print format `[6.0]`, that will contain the current case number.

2. The operator `nocdm` returns the number of cases in the data matrix, or 0 if there is no data matrix.

3. The operator `noc` returns the number of currently selected cases in the data matrix, or 0 if there is no data matrix.

4. The operator `nvar` returns the number of currently defined variables.

5. The operator `bnoc` returns the number of blocks that have been found by the latest use of the `dblock` command.

6. The operator

   ```
   num(x,d)
   ```

   where x and d are (floating point) expressions provides the sequence of numbers:

   $$x + id \quad \text{for} \quad i = 0, 1, 2, \ldots$$

7. The operator

   ```
   exists(Name)
   ```

   requires that `Name` is a syntactically valid matrix, variable, or namelist name. The operator returns 1 if the object denoted by `Name` exists, and otherwise returns 0. In particular,

   ```
   exists()
   ```

   returns 0. This is useful in order to check whether `$n` expressions in macro definitions are actually available when the macro is executed.

For example, `exists($1)` when used inside a macro, will be expanded to `exists(Name)` when `Name` is given for `$1` when invoking the macro, and is expanded into `exists()` when the macro argument `$1` is not used.

Remember that `$n` expressions are substituted by their corresponding names, when given by the user, or substituted by Null-strings. This rule also applies when `$n` expressions are used inside double quotation marks. The only exception is when `$n` expressions are inside single quotation marks. For example,

```
print("$1 ...")
```

will substitute `$1` by a valid name, or by the Null-string; but

```
print("... '$1' ...")
```

will not substitute `$1` by a name or the Null-string.

8. The operator

```
row(expression)
```

returns the row dimension of `expression`, and the operator

```
col(expression)
```

returns the column dimension of `expression` (see

5.2.5.2 Arithmetical Operators

The following arithmetical operators are available, all classified as type 1 operators. In describing these operators, x and y are numerical (integer or floating point) expressions.

$x + y$	addition
$x - y$	subtraction
$x * y$	multiplication
$x \ / \ y$	division (y must not be zero)
$x \ \hat{} \ y$	power of (y must not be negative)
$x \ \% \ y$	modulus operator (x and y must be integer)
$x \ \& \ y$	logical and; result is 1 if both x and y have nonzero value, otherwise the result is zero.
$x \mid y$	logical or; result is 1 if x or y has a nonzero value, otherwise the result is zero.

When an expression contains more than one of these operators, evaluation follows the usual priority rules: $(+, -, |)$, $(*, /, \&)$, $(\hat{}, \%)$ in ascending order. Brackets may be used to control the order of evaluation.

5.2.5.3 Mathematical Operators

The following mathematical operators are available. All arguments are numerical (floating point) expressions.

1. `rnd (`x`)` provides the integer value of x; rounded to the nearest integer. This is identical with

   `sign(`x`) * floor(abs(`x`) + 0.5)`

2. `abs(`x`)` evaluates the absolute value of x.
3. `sign(`x`)` provides the sign of x, i.e., +1 if x is greater than zero, -1 if x is less than zero; and otherwise 0.
4. `floor(`x`)` provides the largest integer not greater than x.
5. `ceil(`x`)` provides the smallest integer not less than x.
6. `tr(`x, a, b`)` truncates x to the interval $[a, b]$, meaning that the result is equal to a if $x < a$, equal to b if $x > b$, and other is equal to x.
7. `min(`$x_1, x_2, \ldots$`)` gives the minimum of its arguments. The operator has a variable number of arguments.
8. `max(`$x_1, x_2, \ldots$`)` gives the maximum of its arguments. The operator has a variable number of arguments.
9. `sqrt(`x`)` gives the square root of x. It is required that $x \geq 0$.
10. `exp(`x`)` evaluates the exponential function (antilogarithm) of x.
11. `eexp(`x`)` evaluates $\exp(x)/(1 + \exp(x))$.
12. `log(`x`)` evaluates the natural logarithm of x. It is required that $x > 0$.
13. `sin(`x`)` evaluates the sine of x.
14. `cos(`x`)` evaluates the cosine of x.
15. `lgam(`x`)` provides the natural logarithm of the gamma function, evaluated at x. It is required that $x > 0$. The algorithm is adopted from Pike and Hill [1966].

16. `digam(`x`)` provides the first derivative of the logarithm of the gamma function, evaluated at x. It is reqired that $x > 0$. The algorithm is adopted from Bernado [1976].

17. `trigam(`x`)` provides second derivative of the logarithm of the gamma function, evaluated at x. It is required that $x > 0$. The algorithm is adopted from Schneider [1978].

18. `icg(`x, q`)` evaluates the incomplete gamma integral, that is,

$$\text{icg}(x, q) \;=\; \frac{1}{\Gamma(q)} \int_0^x t^{q-1} e^{-t} dt, \quad x, q > 0$$

    The algorithm is adopted from Moore [1982]. Note that there is another operator, `icg1`, that also evaluates the incomplete gamma integral, based on an algorithm by Lau [1980]. However, while `icg` can be used with automatic differentiation, this is not possible with `icg1`.

19. `icb(`x, a, b`)` provides the incomplete beta integral, evaluated for x with coefficients given by a and b. It is required that $0 < x < 1$. The algorithm is adopted from Baker ([1992], p. 580).

20. `bc(`n, m`)` returns the binomial coefficient, n over m. n and m must be integers. It is required that $0 \leq m \leq n$. The operator can easily lead to overflows. One should then consider to use the `lgam` operator for the logarithm of the gamma function.

5.2.5.4 Logical Operators

Logical operators to build expressions may have one, two, or three arguments. The arguments are any expressions, the result is a logical expression, that is, only two values are distinguished: a logical expression is true if it evaluates to a nonzero value, otherwise the logical expression is false. The following logical operators are available:

1. `not (E)` returns the logical negation of expression `E`, that is, if `E` is true, `not (E)` returns 0 (false); if `E` is false, `not (E)` returns 1 (true).

2. `and (E1,E2)` returns the logical AND of expressions `E1` and `E2`, that is, if both `E1` and `E2` are true, `and (E1,E2)` returns 1 (true), otherwise it returns 0 (false). Note that it is often more convenient to write `E1 & E2` instead of `and (E1,E2)`.

3. `or (E1,E2)` returns the logical OR of expressions `E1` and `E2`, that is, if `E1` or `E2` is true, `or (E1,E2)` returns 1 (true), otherwise it returns 0 (false). Note that it is often more convenient to write `E1 | E2` instead of `or (E1,E2)`.

4. `eq (E1,E2)` returns 1 (true) if expressions `E1` and `E2` have equal value, otherwise it returns 0 (false).

5. `ne (E1,E2)` returns 1 (true) if expressions `E1` and `E2` do not have equal value, otherwise it returns 0 (false).

6. `lt (E1,E2)` returns 1 (true) if the value of expressions `E1` is less than the value of expression `E2`, otherwise it returns 0 (false).

7. `le (E1,E2)` returns 1 (true) if the value of expressions `E1` is less than, or equal to, the value of expression `E2`, otherwise it returns 0 (false).

8. `gt (E1,E2)` returns 1 (true) if the value of expressions `E1` is greater than the value of expression `E2`, otherwise it returns 0 (false).

9. `ge (E1,E2)` returns 1 (true) if the value of expressions `E1` is greater than, or equal to, the value of expression `E2`, otherwise it returns 0 (false).

10. `if (E,E1,E2)` returns `E1` if the logical expression `E` is true (not equal to zero), and returns `E2` otherwise. This operator, like all others, can

be used recursively, for example,

```
if(E,E1,if(E2,E3,E4))
```

When using the `if` operator for the definition of variables, one can also use another, sometimes more convenient, syntax:

```
if E then E1 else E2
```

is then equivalent to

```
if(E,E1,E2)
```

or more general:

```
if E then E1 else if EE then EE1 else if ... else E2
```

5.2.5.5 Dummy Variables

There are two operators to ease creation of dummy variables with cornered or with centered effects. In describing these operators, `E` is an expression and `J1,J2,...`, and `K1,K2,...`, are integer expressions.

1. The result of

```
E[J1,J2,...]
```

is a cornered effect dummy variable taking the value 1 if `E` has any one of the integer values `J1,J2,...`; otherwise the result is 0. A comma can be substituted by two commas, meaning then the range between two integer values. For instance, `E[J1,,J2]` would result in the value 1 if the expression `E` takes any one of the values in the range from `J1` to `J2`.

2. The result of

```
E[J1,J2,...:K1,K2,...]
```

is a centered dummy variable, taking the value 1 if `E` is equal to one of the integer expressions `J1,J2,...`, taking the value -1 if `E` is equal to one of the integer expressions `K1,K2,...`; and is otherwise 0.

The following box provides some examples.

```
X  X[1]  X[1,7]  X[1,,3]  X[1,3,,7]  X[1:2,3]
-----------------------------------------------
1    1      1        1          1          1
7    0      1        0          1          0
2    0      0        1          0         -1
3    0      0        1          1         -1
1    1      1        1          1          1
```

In order to give sensible results, `E` should be an integer expression. In fact, the expression E is always truncated (not rounded) to an integer expression before making comparisons with the arguments `J1,J2,...`, and `K1,K2,...` If `E` is not a simple expression, it should be enclosed in brackets, for instance:

```
(E1 * (E2 + E3))[...]
```

Dummy variables can also be created with the `ndvar` command, see 2.5.

5.2.5.6 Density Functions

The following operators are available to calculate density functions. In describing these operators, x represents a numerical (floating point) expression.

1. `ndf(`x`)` returns the value of $\phi(x)$, the standard normal density function. The operator can be used for automatic differentiation.

2. `mr(`x`)` returns the reciprocal value of Mill's ratio, that is,

$$\mathrm{mr}(x) = \phi(x)/(1 - \Phi(x))$$

   where ϕ is the density and Φ the distribution function of the standard normal distribution. The algorithm is adopted from Swan [1969].

3. `poisson(`θ, k`)` returns the logarithm of the Poisson distribution, i.e.,

$$\log\left(\frac{\theta^k}{k!\,\exp(\theta)}\right) = k\log(\theta) - \log(\Gamma(k+1)) - \theta$$

   Automatic differentiation is possible only wrt θ.

4. `negbin(`α, γ, k`)` returns the logarithm of the negative binomial distribution, i.e.,

$$\log\left(\frac{(\Gamma(\alpha+k)}{\Gamma(k+1)\,\Gamma(\alpha)}\left(\frac{\gamma}{\alpha+\gamma}\right)^k\left(\frac{\alpha}{\alpha+\gamma}\right)^\alpha\right)$$

   The operator allows for automatic differentiation with respect to its first two arguments, α and γ, see **6.14.2**.

5.2.5.7 Distribution Functions

The following operators evaluate distribution functions. In describing these operators, x represents a floating-point numerical expression.

1. `nd(`x`)` return the value of $\Phi(x)$, i.e., the standard normal distribution function. The algorithm is adopted from Hill [1973]. The operator can be used for automatic differentiation.

2. `ndi(`x`)` returns the value of $\Phi^{-1}(x)$, i.e., the inverse standard normal distribution function. The algorithm is adopted from Hill and Davis [1971]. (This is a replacement for the algorithm from Beasley and Springer [1977] that was used in earlier versions of TDA.) The operator supports automatic differentiation for its first derivative.

3. `td(`x, n`)` returns the value of the t-distribution, evaluated at x, with degress of freedom given by the integer expression n. The algorithm is adopted from Levine [1969].

4. `chd(`x, n`)` returns the value of the χ^2-distribution, evaluated at x, with degrees of freedom given by the integer expression n. The algorithm is adopted from Hill and Pike [1967].

5. `fd(`x, n, m`)` returns the value of the F-distribution, evaluated at x, with degrees of freedom given by the integer expressions n and m. The algorithm is adopted from Dorrer [1968].

6. `bivn(`x, y, ρ`)` returns the value of the bivariate normal distribution with correlation ρ, that is,

$$\texttt{bivn}(x, y, \rho) \;=\; \Phi_2(x, y, \rho) \;=\; \int_{-\infty}^{x} \int_{-\infty}^{y} \phi_2(u, v, \rho)\, dv du$$

where

$$\phi_2(x, y, \rho) = \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\left\{ -\frac{1}{2} \frac{x^2 + y^2 - 2\rho xy}{1-\rho^2} \right\}$$

The algorithm is adopted from Donnelly [1971]. (Values can be compared with tables in Tong [1990].)

The operator allows for automatic differentiation. The derivatives are calculated as follows (see Tong [1980, p 9] for derivatives wrt ρ).

$$\begin{aligned}
\frac{\partial \Phi_2(x,y,\rho)}{\partial x} &= \phi(x)\,\Phi\Big(\frac{y-\rho x}{\sqrt{1-\rho^2}}\Big) \\
\frac{\partial \Phi_2(x,y,\rho)}{\partial \rho} &= \phi_2(x,y,\rho) \\
\frac{\partial^2 \Phi_2(x,y,\rho)}{\partial x \partial x} &= -\phi(x) \\
&\quad \left\{ \frac{\rho}{\sqrt{1-\rho^2}}\,\phi\Big(\frac{y-\rho x}{\sqrt{1-\rho^2}}\Big) + x\,\Phi\Big(\frac{y-\rho x}{\sqrt{1-\rho^2}}\Big) \right\} \\
\frac{\partial^2 \Phi_2(x,y,\rho)}{\partial x \partial y} &= \frac{1}{\sqrt{1-\rho^2}}\,\phi(y)\,\phi\Big(\frac{x-\rho y}{\sqrt{1-\rho^2}}\Big) \\
\frac{\partial^2 \Phi_2(x,y,\rho)}{\partial x \partial \rho} &= -\frac{x-\rho y}{1-\rho^2}\,\phi_2(x,y,\rho) \\
\frac{\partial^2 \Phi_2(x,y,\rho)}{\partial \rho \partial \rho} &= \phi_2(x,y,\rho) \left\{ \frac{1}{1-\rho^2} \left(\frac{x-\rho y}{\sqrt{1-\rho^2}} \frac{y-\rho x}{\sqrt{1-\rho^2}} + \rho \right) \right\}
\end{aligned}$$

ϕ and Φ denote, respectively, the density and distribution function of the standard normal distribution.

7. `mvn`$(\Sigma, k, \epsilon, x_1, \ldots, x_n)$ returns the value of an n-dimensional normal distribution with correlation matrix Σ, that is

$$\frac{1}{(2\pi)^{n/2}\,|\Sigma|^{1/2}} \int_{-\infty}^{x_n} \cdots \int_{-\infty}^{x_1} \exp\Big(-\frac{1}{2}\,x'\Sigma^{-1}x\Big) dx_1 \cdots dx_n$$

where $x = (x_1, \ldots, x_n)'$. The number of dimensions is restricted to $2 \le n \le 20$. The operator expects as its first argument the name of an (n, n) correlation matrix that must be defined with the `mdef` command, see **5.1.2**. Its upper triangle is used for the correlation matrix.

The algorithm is adopted from Drezner [1992] and uses an adaptive Gaussian quadrature method for numerical integration. There are two options, depending on k (integer) and ϵ (floating point). If $2 \le k \le 10$, ϵ is ignored and the algorithm performs one Gaussian quadrature with k points for each dimension of the integral. If $12 \le k \le 20$, the algorithm performs progressive quadratures with

number of points $m = 2, \ldots, k - 10$, until the difference between two successive evaluations of the integral does not exceed ϵ. To check the operator one can use TDA's `calc` command. For tables with values of the multivariate normal integral see Tong [1990].

The operator cannot be used for automatic differentiation. If derivatives are required they must be approximated numerically. For applications, see **6.12.5**.

5.2.5.8 Julian Calendar

Given variables Y (year, four digits), M (month), and D (day), the operator

$$\texttt{jul}\,(Y, M, D)$$

returns the corresponding day in the Julian calendar. On the other hand, given a day J in the Julian calendar, then

$\texttt{july}(J)$ returns the corresponding year
$\texttt{julm}(J)$ returns the corresponding month
$\texttt{juld}(J)$ returns the corresponding day

Consequently,

$$\begin{aligned}\texttt{july}\,(\texttt{jul}(Y, M, D)) &= Y\\ \texttt{julm}\,(\texttt{jul}(Y, M, D)) &= M\\ \texttt{juld}\,(\texttt{jul}(Y, M, D)) &= D\end{aligned}$$

5.2.5.9 String Variables

The following type 1 operators can be used for string variables.

1. Given a string variable `S` the operator

   strlen(S)

   returns the length of the string variable, i.e., the number of columns that are used for each of the strings.

2. Given a string variable `S` the operator

   strv(S)

   tries to convert the strings in `S` into numerical values. If this is not possible because a string does not consist only of digits, the operator returns -1.

3. Given a string variable `S` the operator

   strvp(S,n,m)

   tries to convert the sub-strings from column `n` until, and including, column `m` into numerical values. If this is not possible because a sub-string does not consist only of digits, the operator returns -1.

5.2.6 Type 2 Operators

Type 2 operators are specific in that they require information not only about the current data matrix row. The actual amount of information depends on the mode of data generation, see **2.2**. In record mode (default), type 2 operators are evaluated for all rows in the current data matrix; in block mode, they are evaluated for all rows in a block. This implies some restrictions in using these operators to define expressions.

Note that a variable is assigned type 4 (see **2.1**) if its definition contains type 2 operators, or if its definition refers to at least one other variable that, in turn, involves type 2 operators. Here are some examples of type 2 operators.

```
 X  vmin(X) vmax(X) sum(X) cum(X) cd(X) cdv(X) mean(X) std(X)
-------------------------------------------------------------
 1     1       3      7      1    0.50  0.2857  1.75   0.9574
 1     1       3      7      2    0.50  0.2857  1.75   0.9574
 2     1       3      7      4    0.75  0.5714  1.75   0.9574
 3     1       3      7      7    1.00  1.0000  1.75   0.9574
```

Currently available type 2 operators will be described in the following subsections.

Note that type 2 operators cannot be used for automatic differentiation.

5.2.6.1 General Type 2 Operators

In describing the following general type 2 operators, X denotes the name of a variable that must be available in the currently defined data matrix.

1. `vmin` (X) provides the minimum of the values of the variable X.
2. `vmax` (X) provides the maximum of the values of the variable X.
3. `sum` (X) provides the sum of the values of the variable X.
4. `cum` (X) provides the cumulated values of the variable X.
5. `cd` (X) provides the empirical distribution function of X.
6. `cdv` (X) provides the incomplete first moment of the distribution of variable X, calculated as

$$\text{cdv}(x) = \frac{\sum_{x_i \le x} x_i}{\sum_{i=1}^{n} x_i}$$

7. `mean` (X) provides the mean value of the variable X.
8. `std` (X) provides the standard deviation of the variable X.
9. `ndv` (X) calculates the number of different values occurring in X.
10. `ndv1` (X, A) calculates the number of different values occurring in variable X that are not less than A (any expression).
11. `ndv2` (X, A, B) calculates the number of different values occurring in variable X that are not less than A and not greater than B.
12. `mav` (X, n) calculates a moving average for variable X:

$$\text{mav}(X, n) = \frac{1}{2n+1} \sum_{j=-n}^{n} X_{i+j}$$

n must be a positive integer.

13. The operator `bfa` $(X[\ldots])$ returns the number of times that the dummy variable $X[\ldots]$ (i.e., the argument of the operator) takes a non-zero value in the current block (or the data matrix, if not in block mode).

14. The operator `bfr` $(X[\ldots])$ returns the relative frequency of the occurrence of non-zero values of the dummy variable $X[\ldots]$ in the current block (or the data matrix, if not in block mode).

15. The operator `quant` (X, p) returns the p-quantile of the distribution of the variable X. Calculation is done with the same method als described for the `quant` command.

16. The operator `quant1` (X, Z, p, m) returns the p-quantile of the distribution of X, but X might contain censored values. This is indicated by values of variable Z. If $Z_i = 0$, X_i is assumed to be a censored value; otherwise X_i is treated as uncensored. If the requested quantile cannot be calculated, the operator returns the value m.

17. The operator `vdif` (X, Y) returns an indicator of different values. Given values X_i and Y_i for $i = 1, \ldots, n$, the result of `vdif` (X, Y) for the ith record is 1 if X_i is not equal to any of the values Y_j for $j = 1, \ldots, n$; otherwise the result will be 0. Example:

X	Y	`vdif` (X, Y)
1	3	0
2	2	0
7	8	1
9	1	1
1	8	0

5.2.6.2 Recode Operators

Assume a variable, X, and a list, L, having two columns. The operator

$$Y \;=\; \texttt{recode}\,(X, L)$$

can be used to create a new variable, say Y, such that whenever X has a value occurring in the first column of L, Y gets the corresponding value in the second column of L. L must be the name of a matrix having at least two columns, defined with an `mdef` command, see 5.1.2.

Using the recode operator, if X has a value not occurring in the first column of L, Y gets the value of X. Alternatively, one can use the operator

$$Y \;=\; \texttt{recode}\,(X, L, v)$$

Then, if a value of X does not occur in L, the new value in variable Y will be v.

5.2.6.3 Leads and Lags

The following operators are available to refer to leads and lags of the values of a variable.

1. Given a variable X, `pre(`X`)` provides its values lagged by one data matrix row. X must be the name of a data matrix variable. Note that the `pre` operator is specific in that it is the only operator that can be used in a self-referencing way. In all other cases, when defining a variable, one can only use references to previously defined variables. In contrast, the `pre` operator allows for constructions as

   $$Y = \text{if ne}(X,\text{pre}(X)) \text{ then } Y \text{ else } Y + \text{pre}(X)$$

   These expressions must not contain any other type 2 operators; otherwise one will get a wrong result.

2. Given a variable X, `suc(`X`)` provides its values leaded by one data matrix row. X must be the name of a data matrix variable.

3. Given a variable X, `lag(`X, n`)` provides its values lagged by n data matrix rows if $n < 0$, or leaded by n data matrix rows if $n > 0$. It is possible to use a general expressions instead of n.

The result of these operators is zero if they refer to nonexistent data matrix rows. In block mode, the result is zero if they refer to data matrix rows outside the current block.

5.2.6.4 Sorting and Ranking

The following type 2 operators can be used to sort, or rank, the values of a variable.

1. Given a variable X, `sort(`X`)` creates a new variables that contains the values of X sorted in ascending order.
2. Given a variable X, `snum(`X`)` provides serial numbers based on sorting the values of X in ascending order.
3. Given a variable X, `rank(`X`)` provides rank numbers for the values of `sort(`X`)`.

If the values of the variable X are not unique, the sorting is not stable. In any case, evaluation depends on being in record mode or in block mode. Some examples are shown in the following box.

X	sort(X)	sqrt(abs(sort(X)))	-sort(-X)	snum(X)	rank(X)
1	-3	1.7321	9	3	3.5
7	-3	1.7321	9	8	8.0
2	1	1.0000	7	6	6.0
9	1	1.0000	2	10	9.0
-3	2	1.4142	2	2	1.5
1	2	1.4142	2	4	3.5
2	2	1.4142	1	7	6.0
-3	7	2.6458	1	1	1.5
9	9	3.0000	-3	9	9.0
2	9	3.0000	-3	5	6.0

5.2.6.5 Counting Subsets

The following type 2 operators can be used to count subsets (of cases) of a variable.

1. `cnteq` (X, Y) returns the number of cases where X is equal to Y.
2. `cntne` (X, Y) returns the number of cases where X is not equal to Y.
3. `cntlt` (X, Y) returns the number of cases where X is less than Y.
4. `cntgt` (X, Y) returns the number of cases where X is greater than Y.
5. `cntle` (X, Y) returns the number of cases where X is less than, or equal to, Y.
6. `cntge` (X, Y) returns the number of cases where X is greater than, or equal to, Y.

X and Y are variables (or more general, expressions). Calculating the counts depends on whether the operators are used in record mode or in block mode. In record mode, the counts are based on the number of cases in the currently defined data matrix, in block mode the counts are calculated for each block separately. Some examples, assuming record mode, are shown in the following box.

V1	V2	cnteq	cntne	cntlt	cntgt	cntle	cntge
1	1	2	8	6	2	8	4
7	2	2	8	6	2	8	4
2	3	2	8	6	2	8	4
9	4	2	8	6	2	8	4
-3	5	2	8	6	2	8	4
1	6	2	8	6	2	8	4
2	7	2	8	6	2	8	4
-3	8	2	8	6	2	8	4
9	9	2	8	6	2	8	4
2	10	2	8	6	2	8	4

5.2.6.6 Counting Changes

The following type 2 operators can be used to provide information about changes in the values of a variable.

1. Given a variable X, `change(`X`)` provides a dummy variable taking the value 1 if the current value of X is not equal to its previous value. X is assumed to be not equal to its predecessor if in the first row of the data matrix or of a block.

2. Given a variable X, `cntch(`X`)` provides the number of changes in the values of X, that is, the number of rows where X is not equal to its previous value. X is assumed to be not equal to its predecessor if in the first row of the data matrix or a block.

3. Given a variable X, `ccntch(`X`)` provides the number of changes in the values of X up to, and including, the current row. X is assumed to be not equal to its predecessor if in the first row of the data matrix or a block.

4. Given a variable X, `lagch(`X`)` provides the number of rows, calculated backwards from the current row, where the last change of X occurred. X is assumed to be not equal to its predecessor if in the first row of the data matrix or a block.

Some examples are shown in Box 1.

Box 1 Examples of operators for changes

Record Mode

ID	S	T	change(S)	cntch(S)	ccntch(S)	lagch(S)	lag(T,-lagch(S))
1	1	10	1	7	1	0	10
1	3	15	1	7	2	0	15
1	2	20	1	7	3	0	20
2	2	12	0	7	3	1	20
2	3	18	1	7	4	0	18
3	3	8	0	7	4	1	18
3	1	12	1	7	5	0	12
3	2	15	1	7	6	0	15
3	2	17	0	7	6	1	15
3	2	18	0	7	6	2	15
3	1	19	1	7	7	0	19

Block Mode

ID	S	T	change(S)	cntch(S)	ccntch(S)	lagch(S)	lag(T,-lagch(S))
1	1	10	1	3	1	0	10
1	3	15	1	3	2	0	15
1	2	20	1	3	3	0	20
2	2	12	1	2	1	0	12
2	3	18	1	2	2	0	18
3	3	8	1	4	1	0	8
3	1	12	1	4	2	0	12
3	2	15	1	4	3	0	15
3	2	17	0	4	3	1	15
3	2	18	0	4	3	2	15
3	1	19	1	4	4	0	19

5.2.6.7 Aggregates

The term *aggregate* is used to denote a contiguous sequence of data matrix rows, identified by having the same value of an identification variable. The following operators to evaluate characteristics of aggregates are available.

1. `gcnt` (A) counts the number of cases in each of the aggregates defined by the variable A.
2. `grec` (A) provides a serial number for the records in an aggregate defined by variable A.
3. `gsn` (A) provides a serial number for the aggregates defined by A.
4. `gfirst` (A) provides a dummy variable with value 1 for the first record in each aggregate defined by A.
5. `glast` (A) provides a dummy variable with value 1 for the last record in each aggregate defined by A.
6. `gsum` (X, A) returns the sum of the values of variable X, calculated separately for each aggregate defined by A.
7. `gmean` (X, A) returns the mean value of variable X, calculated separately for each aggregate defined by A.
8. `gstd` (X, A) returns the standard deviation of variable X, calculated separately for each aggregate defined by A.
9. `gmin` (X, A) returns the minimum of variable X, calculated separately for each aggregate defined by A.
10. `gmax` (X, A) returns the maximum of variable X, calculated separately for each aggregate defined by A.
11. `gsort` (X, A) sorts the values of variable X, in ascending order, separately for each aggregate defined by A.
12. `gndv` (X, A) counts the number of different values in variable X, separately for each aggregate defined by A.

Box 1 Examples of operators for aggregates

```
 V  A  gcnt(A)  grec(A)  gsn(A)  gfirst(A)  glast(A)
-------------------------------------------------------
 1  1      3        1       1         1          0
 4  1      3        2       1         0          0
 5  1      3        3       1         0          1
 3  2      1        1       2         1          1
 6  7      2        1       3         1          0
 2  7      2        2       3         0          1

 V  A  gmean(V,A)  gstd(V,A)  gmin(V,A)  gmax(V,A)
-------------------------------------------------
 1  1     3.333       2.082      1.000      5.000
 4  1     3.333       2.082      1.000      5.000
 5  1     3.333       2.082      1.000      5.000
 3  2     3.000       0.000      3.000      3.000
 6  7     4.000       2.828      2.000      6.000
 2  7     4.000       2.828      2.000      6.000
```

13. `gndv1` (X, Y, A) counts the number of different values in variable X that are not less than Y, separately for each aggregate defined by A.

The aggregate consists of all contiguous data matrix rows that have an identical value of the aggregating variable. Evaluation of the operator depends on being in record mode or block mode. In block mode, there are two levels of aggregates: the first level is defined as blocks, the second level is defined by an aggregating variable inside the blocks. Some examples based on record mode are shown in Box 1.

5.2.6.8 String Variables

The following type 2 operators can be used for string variables.

1. Given a string variable `S` the operator

   strsp(S)

   returns a numerical variable that sorts the strings in `S` in ascending order. For strings which are identical the resulting variable gets the same value.

5.2.7 Block Mode Operators

The following type 2 operators assume block mode, see **2.2**.

1. `brec` provides the current case number for the rows in a block; `brec` is 1 for the first row in block, 2 for the second row, and so on.
2. `bnrec` provides the number of cases contained in the current block.
3. `bfirst` provides a dummy variable with value 1 for the first record in the current block.
4. `blast` provides a dummy variable with value 1 for the last record in the current block.
5. `bmin(`X`)` provides the minimum of the variable X in the current block.
6. `bmax(`X`)` provides the maximum of the variable X in the current block.
7. `brd` provides a random number that is equally distributed in the 0–1 interval. For each block a single new random number is drawn. The algorithm is the same as used for the `rd` operator.
8. `bnum` provides the current block number. Block numbers are: $1, \ldots, m$ where m is the number of blocks in the data matrix.

The following box shows some examples.

```
X  Y  brec  bnrec  bfirst  blast  bmin(X)  bmax(X)   brd
-------------------------------------------------------------
1  1    1     3       1       0       1        5     0.0291
4  1    2     3       0       0       1        5     0.0291
5  1    3     3       0       1       1        5     0.0291
3  2    1     1       1       1       3        3     0.9495
6  7    1     2       1       0       2        6     0.0943
2  7    2     2       0       1       2        6     0.0943
```

5.2.8 Operators for Episode Data

There are a few operators that can be used to get information about currently defined episode data. These operators are also called type 3 operators and can only be used in expressions defining type 5 variables in the `edef` command, see **3.3.2**.

1. `sn` provides the current spell number.
2. `org` provides the current origin state.
3. `des` provides the current destination state.
4. `ts` provides the current starting time.
5. `tf` provides the current ending time.
6. There is one further operator, `time`, that can only used when estimating Cox models.

5.2.9 Operators for Sequence Data

Sequence data are defined by sequences of variables, e.g., $Y_1, \ldots, Y_n$, where only nonnegative values are valid and negative values indicate missings, see **3.4.2**.

1. `slen` $(Y_1, , Y_n)$ calculates the sequence length, that is, the number of variables beginning with the first nonnegative value and ending with the last nonnegative value.

2. `glen` $(Y_1, , Y_n)$ calculates the length of internal gaps, that is, the number of variables inside the valid sequence length having nonnegative values. For more information see **3.4.2**.

5.3 Functions

This chapter explains TDA's concept of functions. It contains the following sections.

5.3.1 Syntax for Functions

We consider real-valued functions

$$f(x_1, \ldots, x_n) \; : \; \mathbf{R}^n \longrightarrow \mathbf{R} \tag{1}$$

The function f is an expression depending on the arguments $x_1, \ldots, x_n$. Arguments can be any strings consisting of lower-case letters and digits. They must begin with a lower-case letter and must be different from predefined keywords and operators. Given this convention, one can define functions in the same way as variables, by using a set of arguments, numerical constants and operators. For example,

$$f(x, y) \equiv \texttt{sin(x) + cos(y)}$$

would be a valid function definition depending on two variables. In defining functions one can use most of TDA's type 1 operators as described in 5.2.5. Type 2 operators cannot be used. Also, for if-then-else constructions one has to use the `if` operator. For instance, to define the function

$$f(x) = \begin{cases} 1 & : \quad x \leq 0 \;\text{ or }\; x \geq 1 \\ 0 & : \quad \text{otherwise} \end{cases}$$

one should use

$$f(x) \equiv \texttt{if(le(x,0) + ge(x,1),1,0)}$$

When defining functions one can also refer to data matrix variables. In general, if the definition of a function contains at least one reference to a data matrix variable, the function values are obtained by summing its expression over all cases in the currently active data matrix. For example, assume a data matrix containing the variables `A` and `B` for m cases. It would then be possible to define a function

$$f(x, a, b) \equiv \texttt{x + A * a + B * b}$$

TDA will evaluate this expression as

$$f(x, a, b) = \sum_{i=1}^{m} x + A_i a + B_i b$$

Using this feature, it is quite easy to formulate function expressions for nonlinear regression and ML estimation. However, such expression can easily become quite complex. Therefore, to allow for clearly arranged expressions, functions can also be defined recursively. The basic idea is first to define parts of a complex expression and then using names of these parts for defining more complex expressions. The syntax is as follows:

$f(x_1, x_2, x_3, \ldots) \equiv$

```
t1  =  expression depending on (x1,x2,...),
t2  =  expression depending on (t1,x1,x2,...),
...
fn  =  expression depending on (t1,t2,...,x1,x2,...)
```

`fn` is a reserved keyword and, although it is optional, should be used to define the final function expression. `t1,t2,...` and `x1,x2,...` can be any parameter strings consisting of lower case letters and digits. All parameter string which do occur on the left side of an equal sign will be interpreted as intermediate expressions (the maximum number is 50); all remaining strings will become arguments of the function. These final function arguments are sorted alphabetically to provide an ordering: first argument, second argument, and so on.

Here is a simple example.

$f(x_1, x_2) \equiv$

```
sx1 = sin(x1),
sx2 = sin(x2),
fn = sx1 + sx2
```

`x1` is the first and `x2` is the second argument of the function. As another example consider the log-likelihood of a simple logit model with a dependent 0-1-variable, `Y`, and two independent variables, `A` and `B`. The log-likelihood can be defined as

$f(b_0, b_1, b_2) \equiv$

```
v = b0 + A * b1 + B * b2,
e = exp(v),
fn = Y * log(e / (1 + e)) + (1 - Y) * log(1 / (1 + e))
```

or in a simpler way as

$f(b_0, b_1, b_2) \equiv$

```
e = b0 + A * b1 + B * b2,
fn = Y * e - log(1 + exp(e))
```

The log-likelihood of a corresponding probit model could be written as

$f(b_0, b_1, b_2) \equiv$

```
e = b0 + A * b1 + B * b2,
fn = Y * log(nd(e)) + (1 - Y) * log(1 - nd(e))
```

5.3.2 Automatic Differentiation

Most algorithms for function minimization need first, some also need second derivatives. In principle, there are four different approaches:

1. The user must also provide expressions for derivatives. Of course, this would be very inconvenient and error prone.

2. Derivatives can be approximated numerically. This is not only very time-consuming but also introduces numerical errors.

3. One can try to automatically create expressions for derivatives based on the function expression provided by the user. However, it would be very difficult to implement this approach in the required generality.

4. Finally, a very simple and efficient method is to calculate derivatives, if required, simultaneously with evaluating the expression defining the function. This approach was originally proposed by Wengert [1964], see also Wexler [1987]. For the purpose of function minimization, this approach seems to be the most efficient and is used in TDA.

In general, the implementation of this approach depends on how a program evaluates expressions. In TDA, given an expression, the program first creates some intermediate code for representing the numerical steps required to evaluate the expression. (To see this intermediate code, one can use the `parse` command.) Based on this intermediate code, the expression can be evaluated efficiently for arbitrary arguments. If required, first and second derivatives are calculated while evaluating this intermediate code for the expression.

There are some limitations for defining the function if derivatives are required. The function may contain:

1. Numerical constants and references to data matrix variables.

2. The elementary operators $+$, $-$, $*$, $/$, and $\hat{}$.

3. The operators `exp`, `log`, `abs`, `sqrt`, `sin`, `cos`, `lgam`, `icg`, `eexp`, `ndf`, and `nd`. With restrictions, one can also use `ndi`, `poisson`, and `negbin`.

4. The logical operator `if`. Most other logical operators result in zero

derivatives. Of course, when using the `if` operator, the result is not necessarily a continuously differentiable function.

5. The `int` operator for numerical integration, see 5.4.

In order to check results one can use the `evalf` command, see 5.3.3.

5.3.3 Evaluating Functions

The command for function evaluation is `evalf` with syntax shown in the following box.

```
evalf (
     fmt=...,              print format
     argument1=...,        value of first argument, def. 0
     argument2=...,        value of second argument, def. 0
     ...,                  and so on
) = function;
```

The right-hand side must provide the definition of the function. All other arguments are optional. As default, all arguments have zero value.

1. The `evalf` command only calculates the function value. If using the command `evalf1`, instead of `evalf`, one also gets first derivatives, based on automatic differentiation as explained in **5.3.2**.

2. As a further option, one can use `evalf2`, instead of `evalf`, and will then get the function value and first and second derivatives.

Box 1 Command file `deriv1.cf` and standard output

```
evalf2(
    fmt = 12.6,          # print format
    x = 2,               # set value of x argument
    y = 5,               # and y argument

)   =  x * y^2;          # define function


Output
------

Function definition:
fn      = x*y^2

Arguments: x=2 y=5

   50.000000    function value
   25.000000    gradient x
   20.000000    gradient y
    0.000000    hessian  x x
   10.000000    hessian  y x
    4.000000    hessian  y y
```

Example 1 To illustrate, we use the simple function $f(x, y) = xy^2$. Box 1 shows the command file `deriv1.cf` used to evaluate this function and its derivatives. The command to request first and second derivatives is `evalf2`. Optional parameters are the print format and values for the arguments of the functions. The output shows the function value and values of the derivatives, evaluated for the given arguments.

5.4 Numerical Integration

Several different methods have been proposed in the literature for numerical integration. In general, there is no best method, and the behavior depends strongly on the type of function and the required accuracy. For this reason, TDA offers a choice between five different methods.

1. Method 1 is the `QNG` algorithm, adapted from Piessens et al. [1983].
2. Method 2 is the `QSUB` algorithm adapted from Patterson [1973].
3. Method 3 is the `SNIFF` algorithm adapted from Garribba et al. [1978].
4. Method 4 is the `QTRAP` algorithm adapted from Press et al. [1988, p. 121].
5. Method 5 is the `QSIMP` algorithm adapted from Press et al. [1988, p. 123].

All these algorithms are adaptive, meaning that they try to reach a given error tolerance by a succession of applying some basic integration rules. Let $f(x)$ denote the integrand, a real-valued function depending on a single argument. It is tried to approximate the integral

$$I = \int_a^b f(x)\,\mathrm{d}x$$

for the finite interval $[a, b]$. Given a relative error tolerance, ϵ_r, an algorithm terminates successfully if

$$|\, I_k - I_{k-1} \,| \;\le\; \epsilon_r \,|\, I_k \,| \tag{1}$$

where I_k denotes the kth approximation to I. All algorithms use this relative convergence criterion. Only algorithm 1 (`QNG`) uses additionally an absolute error tolerance, ϵ_a, with the criterion

$$|\, I_k - I_{k-1} \,| \;\le\; \epsilon_a \tag{2}$$

The algorithm terminates when (1) or (2) is satisfied.

The different methods for numerical integration have been implemented in TDA both on the level of commands and on the level of operators, see

5.4.1 Commands for Numerical Integration

For available algorithms see 5.4. Default is method 1 (the `QNG` algorithm) with relative error = 0.0001. To change this default, one can use the `niset` command with syntax shown in the following box. The selection remains valid until changed with another `niset` command.

```
niset (
      rerr=,          relative error, def. rerr=1.e-4
      aerr=,          absolute error, def. aerr=1.e-4
) = n;                method (1 – 5), def. 1.
```

While the main application of these algorithms in TDA is their use for evaluating log-likelihood functions, they can also be used directly with the `int` command. The following box shows the syntax.

```
int (
      ab=a,b,         integration interval, no default
      fmt=,           print format, def. fmt=0.0
) = function;
```

The command requires the definition of a function that depends on a single argument on its right-hand side. Also required is the specification of an integration interval with the `ab` parameter.

Box 1 Illustration of `int` command

```
Command:

int (ab=0,1) = sqrt(x);

Output:

> int(...)=...
-----------------------------------------------------
Numerical integration. Current memory: 132624 bytes.

Function definition:
fn      = sqrt(x)

Function argument: x
Integration interval: 0 -- 1
Method 1 (QNG).
Relative error:  1.00000e-04
Approximation: 0.666667  (successful)
Number of function calls: 43
```

Example 1 As an example, we try to find the value of

$$I = \int_0^1 x^{1/2}\,\mathrm{d}x$$

which should be equal to 2/3. When using the default method 1, the command is simply

```
int(ab=0,1) = sqrt(x),
```

Box 1 shows this command and its standard output. The command file in the example archive is `int1.cf`.

5.4.2 Operators for Numerical Integration

In order to allow for numerical integration as part of expressions, there are some operators. For integration over a finite interval one can use the operators

```
int (a,b,f(t))
```

where `a`, `b` and `f(t)` are expressions, possibly depending on further arguments. This operator evaluates the integral

$$I = \int_a^b f(t)\,\mathrm{d}t$$

using the currently selected integration method (see the `niset` command in 5.4.1). `t` is a reserved keyword and used to denote the integration variable. If the expression defining `f(t)` does not contain `t`, TDA assumes a constant function and simply returns its value multiplied with $(b-a)$.

Example 1 To replicate the example given in 5.4.1 with the `int` operator, one can use the command

```
evalf = int (0,1,sqrt(t));
```

The command file in the example archive is `int2.cf`.

While it is not possible to use the `int` operator recursively, it can be used as part of more complex expression, in particular as part of function definitions. In general, `a`, `b`, and `f(t)` can be functions depending on arguments. For example, assuming that

$$a = a(x_1,\ldots,x_n), \quad b = b(x_1,\ldots,x_n), \quad f(t) = f(x_1,\ldots,x_n,t),$$

the resulting integral would be the function

$$I(x_1,\ldots,x_n) = \int_{a(x_1,\ldots,x_n)}^{b(x_1,\ldots,x_n)} f(x_1,\ldots,x_n;t)\,\mathrm{d}t$$

If required, first and second derivatives are calculated using automatic differentiation, see 5.3.2. This allows, for example, to define log-likelihood

expression for transition rate models by only using analytical expressions for the transition rate. For calculating first derivatives, TDA uses the formula:

$$\begin{aligned}
&\frac{\partial I(x_1,\ldots,x_n)}{\partial x_i} = \\
&\quad \frac{\partial b(x_1,\ldots,x_n)}{\partial x_i} f(x_1,\ldots,x_n;b) - \\
&\quad \frac{\partial a(x_1,\ldots,x_n)}{\partial x_i} f(x_1,\ldots,x_n;a) + \\
&\quad \int_{a(x_1,\ldots,x_n)}^{b(x_1,\ldots,x_n)} \frac{\partial f(x_1,\ldots,x_n;t)}{\partial x_i}\,\mathrm{d}t
\end{aligned}$$

For calculating second derivatives, TDA uses the formula:

$$\begin{aligned}
&\frac{\partial^2 I(x_1,\ldots,x_n)}{\partial x_i\,\partial x_j} = \\
&\quad \frac{\partial^2 b(x_1,\ldots,x_n)}{\partial x_i\,\partial x_j} f(x_1,\ldots,x_n;b) + \\
&\quad \frac{\partial b(x_1,\ldots,x_n)}{\partial x_i} \frac{\partial f(x_1,\ldots,x_n;b)}{\partial x_j} + \\
&\quad \frac{\partial b(x_1,\ldots,x_n)}{\partial x_j} \frac{\partial f(x_1,\ldots,x_n;b)}{\partial x_i} - \\
&\quad \frac{\partial^2 a(x_1,\ldots,x_n)}{\partial x_i\,\partial x_j} f(x_1,\ldots,x_n;a) - \\
&\quad \frac{\partial a(x_1,\ldots,x_n)}{\partial x_i} \frac{\partial f(x_1,\ldots,x_n;a)}{\partial x_j} - \\
&\quad \frac{\partial a(x_1,\ldots,x_n)}{\partial x_j} \frac{\partial f(x_1,\ldots,x_n;a)}{\partial x_i} + \\
&\quad \int_{a(x_1,\ldots,x_n)}^{b(x_1,\ldots,x_n)} \frac{\partial^2 f(x_1,\ldots,x_n;t)}{\partial x_i\,\partial x_j}\,\mathrm{d}t
\end{aligned}$$

TDA recognizes constant expressions with zero derivatives and skips the corresponding calculations. However, evaluating first and second

Box 1 Example command file `int3.cf` and standard output

```
evalf2(
    x = 0,
    y = 2,
    z = 1.5,

) = int (x,y * z,2 * z^2);

Output:

Function definition:
fn      = int(x,y*z,2*z^2)

Arguments: x=0 y=2 z=1.5

     13.5000    function value
     -4.5000    gradient x
      6.7500    gradient y
     27.0000    gradient z
      0.0000    hessian  x x
      0.0000    hessian  y x
      0.0000    hessian  y y
     -6.0000    hessian  z x
     13.5000    hessian  z y
     36.0000    hessian  z z
```

derivatives of expressions containing the `int` operator can be very time-consuming.

Limitations. One should be aware of an important limitation. The integration variable, `t`, can only be used inside the `int` operator.

Example 2 To illustrate, command file `int3.cf` in Box 1 evaluates the integral

$$I(x, y, z) = \int_x^{yz} 2\,z^2\,\mathrm{d}t$$

and its derivatives. In this example, since the function does not depend on the integration parameter `t`, TDA has actually not used any of the specific subroutines for numerical integration. It would do, however, if the function definition would be changed into

```
int (x,y * z,2 * z^ 2 + 0 * t);
```

Of course, the result should be the same. See command file `int3a.cf` in the example archive.

5.5 Smoothing Procedures

This chapter describes some general-purpose smoothing procedures. For a useful introduction see Goodall [1990]. Further approaches will be discussed in the part on statistical procedures.

5.5.1 Moving Averages

Assume a variable, Y, with values $y_1, \ldots, y_n$. Also assume a sequence of weights: $w_i \geq 0$, for $i = 0, \ldots, s$. We can then defined a smoothed variable, Y^*, by

$$y_i^* := \sum_{k=-s}^{s} w_k y_{i+k}$$

This is called a *moving average* of the original variable, Y. Of course, one will normally use weights having the property

$$\sum_{k=-s}^{s} w_k = 1$$

One needs to specify how to deal with the two ends of the sequence. There are two possibilities.

1. One can restrict the calculation of smoothed values to the range $i = s+1, \ldots, n-s$. This is sometimes called the *copy-on end-value rule*.

2. Alternatively, one can use the end values as often as they are required, meaning that we introduce

$$y_{1-i} := y_1 \quad \text{for} \quad i = 1, \ldots, s$$
$$y_{n+i} := y_n \quad \text{for} \quad i = 1, \ldots, s$$

This is sometimes called the *replicate end-value rule*.

TDA's command for calculating moving averages is called `sma`, the syntax is shown in Box 1. The `gss` parameter must be used to specify weights in the following order:

$$\text{gss} = w_0, w_1, \ldots, w_s,$$

The command calculates a sequence of smoothed values for each variable specified in `varlist` on the right-hand side. The result is written into the standard output. Alternatively, one can specify an output file with the `df` parameter. As an option, one can specify a repeat factor with the `r` parameter meaning that the smoothing procedure is repeated r times.

Box 1 Syntax for `sma` command

```
sma(
      gss=...,          specification of weights
      opt=...,          treatment of end values, def. 1
                        1 = copy-on end-values
                        2 = replicate end values
      r=...,            repeat factor, def. 1
      fmt=...,          print format, def. 10.4
      df=...,           optional output file
      sel=...,          optional case selection
) = varlist;
```

Box 2 Command file `ds3.cf`

```
nvar(
    dfile = ds3.dat,  # data file
    Y = c1,
);
sma(
    gss = 0.5,0.25,
    df  = df,
) = Y;
sma(
    gss = 0.5,0.25,
    opt = 2,            # replicate end values
    dfa = df,           # append
) = Y;
sma(
    gss = 0.5,0.25,
    opt = 2,            # replicate end values
    r = 2,              # repeat
    dfa = df,           # append
) = Y;
```

Example 1 To illustrate, we use the data

$$Y = 1, 3, 7, 6, 6$$

contained in data file `ds3.dat`. The command file, `ds3.cf`, is shown in Box 2. The first `sma` command uses the default copy-on end-value rule. The second command uses the alternative replicate end-value rule. The third command requests that the values are smoothed two times. The resulting output file is shown in Box 3.

Box 3 Output file, `df`, created by `ds3.cf`

```
# Var: Y
  0      1     1.0000     1.0000
  0      2     3.0000     3.5000
  0      3     7.0000     5.7500
  0      4     6.0000     6.2500
  0      5     6.0000     6.0000
# Var: Y
  0      1     1.0000     1.5000
  0      2     3.0000     3.5000
  0      3     7.0000     5.7500
  0      4     6.0000     6.2500
  0      5     6.0000     6.0000
# Var: Y
  0      1     1.0000     2.0000
  0      2     3.0000     3.5625
  0      3     7.0000     5.3125
  0      4     6.0000     6.0625
  0      5     6.0000     6.0625
```

The `sma` Operator It is possible to use the `sma` procedure directly as a type 2 operator. The syntax is:

```
VName = sma[gss=...,r=...,opt=...](Y),
```

where it is assumed that `Y` is the name of an already existing variable. The `sma` operator creates a new variable, `VName`, containing the smoothed values. The `gss` parameter must be used to specify weights. The other parameters are optional.

5.5.2 Running Median Smoothers

A useful smoothing procedure is based on running medians, see Goodall [1990]. The TDA command is `smd` with syntax shown in Box 1. The `sm` parameter must be used to specify a sequence of operations. The syntax is

`sm=[`$c_1\ c_2\ c_3$ `...],`

where each c_i is one of the following characters:

$$c_i = \begin{cases} 2 & \text{2-median} \\ 3 & \text{3-median} \\ 4 & \text{4-median} \\ 5 & \text{5-median} \\ H & \text{hanning} \\ R & \text{replicate until no more changes} \\ S & \text{splitting} \\ t & \text{combine smooth and rough} \end{cases}$$

There are some restrictions in specifying a sequence of these characters. The character R can only be used if it comes after a 3 or 5. The t character, if used, must be the last character in the sequence and must have another character as predecessor. Each character represents an operation; the ordering is from left to right. Given a variable, Y, with values $y_1, \ldots, y_n$, the operations performed are

$$y_i \xrightarrow{c_1} y'_i \xrightarrow{c_2} y''_i \xrightarrow{c_3} \ldots$$

meaning that each operation uses the result from the previous operation as its input.

1. If $c_i = 2$, the operation consists in applying an alternate 2-median:

$$y'_i = \begin{cases} \text{if} \quad r = 0 : & \text{med}(y_{i-1}, y_i) \quad \text{for } i = 2, \ldots, n \\ \text{if} \quad r = 1 : & \text{med}(y_i, y_{i+1}) \quad \text{for } i = 1, \ldots, n-1 \end{cases}$$

At the beginning, $r = 0$. Then, if the 2-median (or a 4-median) operation is finished, r gets the value 1. Then, if the second 2-median or 4-median operation is finished, r gets again the value 0, and so on.

Box 1 Syntax for `smd` command

```
smd (
      sm=[...],          sequence of operations
      opt=...,           treatment of end values, def. 1
                         1 = copy-on end-values
                         2 = replicate end values
                         3 = E rule
      fmt=...,           print format, def. 10.4
      df=...,            optional output file
      sel=...,           optional case selection
) = varlist;
```

2. If $c_i = 3$, the operation consists in applying a 3-median:

$$y'_i = \text{med}(y_{i-1}, y_i, y_{i+1})$$

3. If $c_i = 4$, the operation consists in applying an alternate 4-median:

$$y'_i = \begin{cases} \text{if} \quad r = 0: & \text{med}(y_{i-2}, y_{i-1}, y_i, y_{i+1}) \quad \text{for } i = 3, \ldots, n-1 \\ \text{if} \quad r = 1: & \text{med}(y_{i-1}, y_i, y_{i+1}, y_{i+2}) \quad \text{for } i = 2, \ldots, n-2 \end{cases}$$

4. If $c_i = 5$, the operation consists in applying a 5-median:

$$\begin{aligned} y'_i &= \text{med}(y_{i-2}, y_{i-1}, y_i, y_{i+1}, y_{i+2}) \quad \text{for} \quad i = 3, \ldots, n-2 \\ y'_2 &= \text{med}(y_1, y_2, y_3) \\ y'_{n-1} &= \text{med}(y_{n-2}, y_{n-1}, y_n) \end{aligned}$$

5. If $c_i = H$, the operation consists in calculating the moving average:

$$y'_i = 0.25y_{i-1} + 0.5y_i + 0.25y_{i+1} \quad \text{for} \quad i = 2, \ldots, n-1$$

6. If $c_i = S$, the operation consists in a splitting procedure applied to y_i and y_{i+1}. This is only done if $y_i = y_{i+1}$ and $3 \leq i \leq n-3$. The splitting procedure sets:

$$\begin{aligned} y'_i &= \text{med}(y_i, y_{i-1}, 3y_{i-1} - 2y_{i-2}) \\ y'_{i+1} &= \text{med}(y_{i+1}, y_{i+2}, 3y_{i+2} - 2y_{i+3}) \end{aligned}$$

Box 2 Command file `ds4.cf`

```
nvar(
    noc = 20,       # create random data
    C = case,
    A = rd,
    M1 = smd[sm=[3R]](A),
    M2 = smd[sm=[4253Ht]](A),
    M3 = smd[sm=[3RSSHt]](A),
);
psfile = ds4.ps;
psetup(
    pxa = 0,21,
    pya = 0,1,
    pxlen =  90,
    pylen =  50,
);
plxa(sc=1);
plya(sc=1,ic=10);
plot(lt=0,s=5,fs=1.0) = C,M1;
plot=C,M1;
plot(lt=5) = C,M2;
plot(lt=8) = C,M3;
```

7. If $c_i = R$, this is not a specific operation but the previous operation (which must be a 3-median or a 5-median) is repeated until no more changes occur in the smoothed values.

8. If the last character in the sequence of operations specified with the `sm` parameter is t, smoothed and rough components are combined as follows:

$$y'_i = S(y_i) + S(y_i - S(y_i))$$

where y_i refers to the original values of Y and $S(y)$ means the result of applying the whole sequence of operations to y.

By default (`opt=1`), the `smd` command uses the copy-on end-value rule. If `opt=2`, the replicate end-value rule is used, but only for the H operation. If `opt=3`, and if the operation is a 3-median or 5-median, the algorithm additionally performs:

$$\begin{aligned} y'_1 &= \text{med}(y_1, y'_2, 3y'_2 - 2y'_3) \\ y'_n &= \text{med}(y_n, y'_{n-1}, 3y'_{n-1} - 2y'_{n-2}) \end{aligned}$$

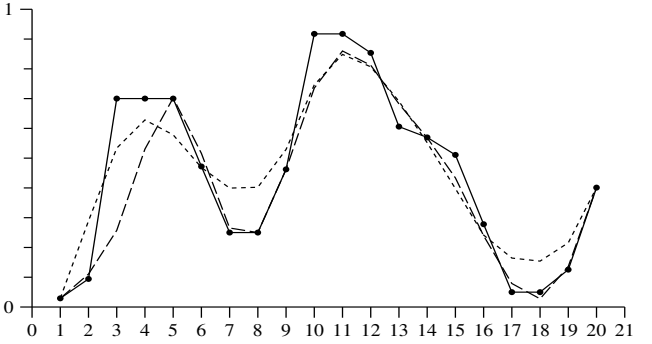


Figure 1 PostScript plot, created with command file `ds4.cf`, that illustrates the `smd` operator.

The `smd` Operator It is possible to use the `smd` procedure directly as a type 2 operator. The syntax is:

```
VName = smd[sm=[...],opt=...](Y),
```

where it is assumed that `Y` is the name of an already existing variable. The `smd` operator creates a new variable, `VName`, containing the smoothed values. The `sm` parameter must be used to specify a sequence of operations in the same way as has been explained above for the `smd` command.

Example 1 To illustrate, command file `ds4.cf`, shown in Box 2, first creates 20 random numbers and then applies the `smd` operator with three different sequences of smoothing operations. It finally generates a PostScript plot as shown in Figure 1.

5.5.3 Smoothing Splines

Dierckx [1975] has published a useful algorithm that calculates smoothing splines (for a general introduction to splines see Boor [1978]). This algorithm has been implemented in TDA. The command is `spl` with syntax shown in Box 1. The command requires two or three variables on the right-hand side. If there are three variables, the values of the third one are interpreted as weights.

These variables provide the input data (x_i, y_i), or if weights are given by a third variable, then (x_i, y_i, w_i), for $i = 1, \ldots, n$. By default, $w_i = 1$ if no weight variable is specified. Weights must be strictly positive. The command then calculates a spline function, $s(x)$, with degree $k \geq 2$, the default is $k = 2$. Higher degrees can be specified with the `deg` parameter. The number of data points, n, must be at least $2k$ and the x_i values must be strictly increasing. The `spl` command tries to achieve this by sorting the input data in ascending order. It will exit with an error message if there are two, or more, equal values in the `X` variable.

The resulting spline also depends on the smoothing factor, σ, that can be specified with the `sig` parameter. By default, $\sigma = 0$ and the algorithm calculates an interpolating spline. If $\sigma > 0$, a smoothing spline is calculated, meaning the smoothest spline function satisfying

$$\sum_{i+1}^{n} w_i(y_i - s(x_i))^2 \leq \sigma$$

If σ is much larger than 0, the result will be an OLS approximation with a polynomial of degree k.

The algorithm requires storage space for a maximum number of knots that cannot be determined in advance. In general, the number of knots actually used, n_k, is

$$3k \leq n_k \leq n + k + 1$$

The default maximum number of knots is

$$\max\{3k + 1, n/2 + 1\}$$

If this is not sufficient, the user can specify another maximum with the `max` parameter. Furthermore, the algorithm requires some iterative

Box 1 Syntax for `spl` command

```
spl (
      sig=...,          smoothing factor, def. 0.0
      deg=...,          degree of spline, def. 2
      max=...,          max number of knots, def. see text
      mxit=...,         max number of iterations, def. 10
      fmt=...,          print format, def. 10.4
      df=...,           output file
      rx=a(d)b,         range for interpolation
) = X,Y [,W];
```

procedures and may not be able to reach convergence. The maximum number of iterations can be specified with the `mxit` parameter. The default value is 10 and should be sufficient for most applications.

There are two options of requesting an output. Both require that an output file is specified with the `df` parameter. With the first option (default), the output file will contain n records, an index number followed by

$$x_i, y_i, \frac{d\, s^j(x)}{d^j\, x} \; (j = 0, \ldots, k)$$

possibly followed by a column containing the weights. So one not only gets the smoothed values, but also derivatives up to the number of degrees.

Alternatively, one can use the `rx` parameter to specify a range of values for interpolation. The syntax is

rx = a (d) b,

where it is required that $a < b$ and $d > 0$. It is then tried to find values of $s(x)$ and its derivatives for the values

$$a + id \quad \text{for} \;\; i = 0, 1, 2, \ldots \;\; \text{until} \; b$$

It may happen that the algorithm is not able to calculate the function for the whole interval. The largest interval that can be used will be shown in the standard output.

Box 2 Command file `ds5.cf`

```
nvar(
    noc = 100,
    X[10.4] = case / 10,
    Z[10.4] = sin(X),
    Y[10.4] = Z + rd(-0.2,0.2),
);
psfile = ds5.ps;
psetup(
    pxa = 0,10,
    pya = -1.5,1.5,
    pxlen =  90,
    pylen =  50,
);
plxa(sc=1);
plya(sc=0.5,ic=5,fmt=4.1);
plot(lt=0,s=2,fs=1.0) = X,Y;
plot(lt=5) = X,Z;

spl(
    sig = 1,
    df = df,
) = X,Y;
clear;
nvar(
    dfile = df,
    X[10.4] = c2,
    Y[10.4] = c4,
);
plot = X,Y;
```

Example 1 To illustrate, command file `ds5.cf`, shown in Box 2, first creates some random data and then uses the `spl` operator to calculate a smoothing spline with smoothing factor $\sigma = 1$. It finally generates a PostScript plot shown in Figure 1. The solid line represents the spline function.

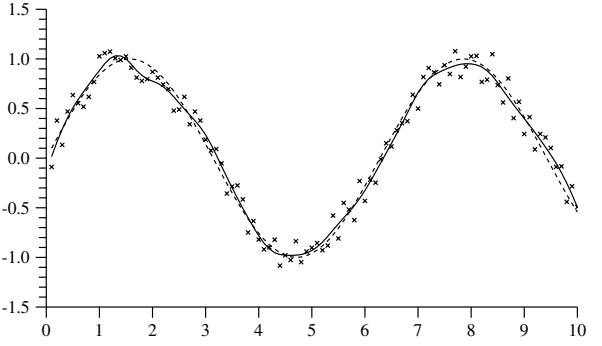


Figure 1 PostScript plot, created with command file `ds5.cf`, that illustrates the `spl` command.

5.6 Function Minimization

This chapter describes a command, and some algorithms, that can be used to find a local minimum of a function. The same algorithms can be used for nonlinear regression and maximum likelihood estimation. This will be discussed in different sections of the manual. Here we have the following sections.

5.6.1 The `fmin` Command

The command for function minimization is `fmin` with syntax

```
fmin (parameter) = function;
```

where `function` is a function as explained in 5.3.1. Optional parameters are shown in Box 1.

1. The `mina` parameter can be used to select a minimization algorithm. For available algorithms and their features see 5.6.2. The default algorithm is Newton (I) and requires first and second derivatives. For most functions, these derivatives will be available via automatic differentiation. Otherwise one should select an algorithm that only needs first derivatives, or no derivatives at all. As a special option for algorithms 7 and 8, one can use the `dopt=1` parameter to request numerical approximation of derivatives.

2. If $f(\theta)$ is the function given as input to the `fmin` command, it actually minimizes $\sigma f(\theta)$ where σ is some scaling factor. This scaling factor can be specified with the parameter

```
dscal = σ,
```

Scaling is helpful to reduce the probability of numerical overflows and underflows. However, the scaling factor σ only influences the global range of the function. There is no automatic procedure for also scaling the individual parameters in the function. The user should therefore take care that all variables have similar magnitudes.

3. The `mxit` parameter can be used to specify a maximal number of iterations. For default values, and also for stopping criteria and convergence tolerances, see 5.6.2.

4. The `prot` parameter can be used to request a protocol file. For more information see 5.6.3.

5. The `xp` and `dsv` parameters can be used to define starting values. How to use these parameters will be explained in 5.6.4.

Box 1 Syntax for `fmin` command

```
fmin(
   mina=...,      minimization algorithm, def. 5
   dopt=...,      1 for numerical approx of derivatives, def. 0
   dscal=...,     scaling factor for function, def. 1
   mxit=...,      max number of iterations
   mxitl=...,     max number of line search iterations, def. 50
   crit=          convergence criterion
   tolg=...,      tolerance: norm of gradient, def. 10^-6
   tolf=...,      tolerance: change of function value, def. 10^-12
   tolp=...,      tolerance: change of parameters, def. 10^-4
   tolv=...,      tolerance: variance of function values, def. 10^-10
   tols=...,      tolerance: reduction factor, def. 10^-4
   tolsg=...,     tolerance: scaled gradient, def. 10^-5
   tolsp=...,     tolerance: scaled parameter change, def. 10^-8
   slen=...,      step length, def. 1
   sred=...,      reduction factor, def. 0.5
   smin=...,      minimum of step size, def. 10^-10
   sst=...,       option for evaluation of derivatives, def. 0
   prot=...,      (or prot1) name of protocol file
   pfmt=...,      print format for protocol file, def. -19.11
   xp=...,        starting values
   dsv=...,       input file with starting values
   ppar=...,      output file with estimated parameters
   tfmt=...,      print format for parameters, def. 10.4
   ccov=...,      type of covariance matrix
   pcov=...,      output file with covariance matrix
   mfmt=...,      print format for covariance matrix, def. 12.4
   mplog=...,     write function minimum into matrix
   mppar=...,     write minimizing arguments into matrix
   mpcov=...,     write covariance matrix into matrix
   mpgrad=...,    write gradients into matrix
   pres=...,      additional output file
   mfmt=...,      print format for pres option, def. 12.4
   v=...,         varlist for pres option
) = function;
```

6. The `ppar` parameter can be used to request an additional output file that will contain the estimated parameters (and standard errors, if any). The print format can be controlled with the `tfmt` parameter.

Box 2 Command file `fmin1.cf` and output

```
fmin = (x - 5)^2 + (y + 6)^2;

Output:

Function definition:
fn      = (x-5)^2+(y+6)^2

Function minimization.
Algorithm 5: Newton (I)

Number of model parameters: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: 1

Idx  Parameter  Starting value
  1  x               0.0000
  2  y               0.0000

Convergence reached in 2 iterations.
Number of function evaluations: 2 (2,2)

Minimum of function: 0
Last absolute change of function value: 1
Last relative change in parameters: 1

Idx  Parameter       Value      Error    Value/E  Signif
-------------------------------------------------------
  1  x              5.0000
  2  y             -6.0000

Function (starting values):    61.0000
Function (final estimates):     0.0000
```

7. A covariance matrix is only calculated if the function refers to at least one data matrix variable and the number of cases exceeds the number of function arguments to provide at least one degree of freedom. The `ccov` parameter can be used to select a calculation method for the covariance matrix, and the `pcov` parameter can be used to write the covariance matrix into an output file. For more information about covariance matrix calculation see **5.6.5**.

8. The `pres` parameter can be used to request an output file containing individual function values for all data matrix cases. If the function definition refers to data matrix variables, and if the minimization procedure finished successfully, the `pres` parameter creates the specified output file containing: a case number for the data matrix rows and the function value evaluated separately for each data matrix row. The print format for the function values can be controlled with the `fmt` parameter, default is `fmt=12.4`. In addition, one can use the parameter

```
v = varlist,
```

to add the variables specified in `varlist` to the output file.

Example 1 To illustrate the `fmin` command, we begin with a simple example without reference to a data matrix. The function is

$$f(x, y) = (x - 5)^2 + (y + 6)^2$$

Box 2 shows the command file, `fmin1.cf`, and the standard output from its `fmin` command. Since the function definition does not refer to data matrix variables, a covariance matrix is not calculated and there are no standard errors.

Example 2 To illustrate reference to data matrix variables we calculate the mean value of a variable $A_i = 1, \ldots, 101$ by minimizing

$$f(x) = \sum_{i=1}^{101} (x - A_i)^2$$

Box 3 shows the command file, `fmin2.cf`, and the standard output from its `fmin` command. The covariance matrix is written to `cov`; the protocol file will give information about intermediate calculations. The `pres` parameter in the command file creates an additional output file, `res`, containing three variables. First, the case number. Second, the function value for each case; in this example, it is simply $(51 - i)^2$. The third variable is `A`, specified with the `v` parameter.

Box 3 Command file `fmin2.cf` and part of output

```
nvar(               # define data: A = 1,...,101
    A = case,
    noc = 101,
);
fmin (
    prot1 = p,      # protocol file
    pcov = cov,     # covariance matrix
    pres = res,     # print individual function values to res
    v = A,          # add variable A

) = fn = (x - A)^2; # function definition


Part of standard output
-----------------------
Function definition:
fn      = (x-A)^2

Function minimization.
Algorithm 5: Newton (I)

Number of model parameters: 1
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: 1

Idx  Parameter  Starting value
  1  x               0.0000

Convergence reached in 2 iterations.
Number of function evaluations: 3 (3,3)

Minimum of function: 85850
Norm of final gradient vector: 4.9738e-13
Last absolute change of function value: 0.753695
Last relative change in parameters: 1

Idx  Parameter       Value      Error    Value/E  Signif
-------------------------------------------------------
  1  x             51.0000     0.0707   721.2489  1.0000

Function (starting values): 348551.0000
Function (final estimates): 85850.0000
```

5.6.2 Minimization Algorithms

Many different algorithms have been proposed in the literature for function minimization. Our basic references are McCormick [1983] and Fletcher [1987]. For a short overview, see also Harley [1986]. TDA supports the methods shown in the following box.

1	Direct search
2	Simplex algorithm
3	Method of conjugate gradients
4	BFGS algorithm
5	Newton algorithm (I)
6	Newton algorithm (II)
7	CES (quadratic model)
8	CES (tensor model)

Direct search. The most simple, but least efficient method is *direct search*. TDA's version of this method is based on an algorithm given by Kaupe [1963] taking into account some modifications proposed by Bell and Pike [1966], Tomlin and Smith [1969], and Smith [1969]. The algorithm does not need any derivatives, selection is with `mina=1`.

The algorithm is controlled by two parameters, a step length and a reduction factor. They can be changed with the `slen` and `sred` parameters, respectively. The step length, multiplied with the absolute values of the parameter components, determines how the parameter space is searched; the default step length is 1. The reduction factor is used for a proportional reduction of the step length in each iteration, the default value is 0.5. The parameters can be changed with the options

```
slen = new_step_length,
sred = new_reduction_factor,
```

The algorithm terminates when the maximum number of iterations (default 100) is reached, or when the reduction factor is less than a given tolerance that can be controlled with the option

```
tols = minimum_of_reduction_factor,
```

The default is `tols` = 10^{-4}.

Simplex method. Another algorithm that does not need derivatives but is more powerful than direct search is based on a simplex procedure, see Nelder and Mead [1965]. TDA's version of this algorithm is adopted from O'Neill [1974] and can be selected with `mina=2`.

The size of the initial simplex can be controlled with the `slen` option, default value is 1. The algorithm terminates if the maximum number of iterations is reached, default is 100, or if the variance of the function values evaluated at the simplex vertices is less than a given tolerance that can be controlled with the parameter

```
tolv = variance_of_function_values,
```

The default value is `tolv` = 10^{-10}.

Both algorithms, direct search algorithm and the simplex procedure, are not recommended in most situations but may sometimes be helpful to find starting values for other methods.

Conjugate gradients and BFGS. There are two variants of Quasi-Newton methods, a method of conjugate gradients and the BFGS algorithm. The implementation of these algorithms in TDA is based on Shanno and Phua [1986]. Both algorithms need first derivatives; the method of conjugate gradients can be selected with `mina=3`, the BFGS method with `mina=4`.

For both algorithms, the convergence check is based on the Euclidean norm of the gradient of the function. Iterations end if this norm becomes less than a given tolerance. The default value of this tolerance is 10^{-6}, but can be changed with the parameter

```
tolg = norm_of_gradient,
```

The default maximum number of iterations for these algorithms is 100, but can be changed with the `mxit` option.

Two more parameters can be used to control the line search. The default maximum number, 50, of sub-iterations during line search can be changed with the parameter

```
mxitl = maximum_number_of_line_search_iterations,
```

and the minimum step length, default 10^{-10}, can be changed with the parameter

```
smin = minimum_step_length,
```

Both algorithms are quite reliable and can be used with all of TDA's models. Normally one would prefer the Newton method discussed below. However, this method may fail due to badly chosen starting values or because calculation of second derivatives is inexact. The method of conjugate gradients or the BFGS algorithm is then a good alternative. With most models implemented in TDA, the BFGS seems to be more efficient than the method of conjugate gradients.

CES algorithms. Recently, Chow, Eskow and Schnabel [1994] have published an algorithm based on tensor methods. Instead of approximating the function by a quadratic model, their algorithm uses a fourth-order model. At least in certain applications, this seems to be more efficient.

The TDA implementation of this method is adapted from the code published by Chow, Eskow and Schnabel [1994]. There are actually two methods, one based on a quadratic model, the other one based on the tensor model. The first method, called `CES (quadratic model)`, can be selected with `mina=7`, the second method, called `CES (tensor model)`, can be selected with `mina=8`.

Convergence check is based on two criteria. First, the algorithm checks the size of a scaled gradient and assumes convergence if

$$\max_i \left\{ \frac{|g_f(\theta_i)|\ \max_j\{\theta_j\}}{\max\{f(\theta), 1\}} \right\} \ \leq\ \texttt{TOLSG}$$

$\theta = (\theta_1, \ldots, \theta_n)$ is the parameter vector, $f(\theta)$ the function value, and $g_f(\theta)$ the gradient. `TOLSG` is the tolerance with default value 1.e-5; this value can be changed with the `tolsg` parameter. A second check is based on changes of the parameter vector. Convergence is then assumed if

$$\max_i \left\{ \frac{|\theta_i - \theta_i^*|}{\max\{\theta_i, 1\}} \right\} \ \leq\ \texttt{TOLSP}$$

where θ^* is the parameter vector from the previous iteration. `TOLSP` is the tolerance with default value 1.e-8; this value can be changed with the `tolsg` parameter. The maximum number of iterations can be controlled with the `mxit` parameter, default is `mxit=20`.

Newton methods. If first and second derivatives are available, a very efficient minimization method for well-behaved functions is the Newton algorithm. The algorithm requires that the function is well-behaved in at least a neighborhood of its minimum. Furthermore, one needs appropriate starting values already near to the minimum. In particular, the

Hessian matrix must be positive definite for all parameter values during the iteration process.

Consequently, for practical implementation, the Newton method must be modified to cope with situations where the Hessian is not positive definite. Many different proposals exist. For implementation in TDA, we have chosen two options. The first (default) option uses simply the gradient if the Hessian is not positive definite.[1] The second, more sophisticated modification follows a proposal of McCormick [1983, p. 166].

To explain the algorithm we shall use the following terminology. $f(\theta)$ is the function to be minimized. $g_f(\theta)$ is the gradient, i.e. the (column) vector of first derivatives. $h_f(\theta)$ is the Hessian matrix, i.e. the matrix of second derivatives. The index k is used to denote iterations.

Then, having some start vector, θ_k, for the kth iteration step, the algorithm tries to find a better estimate, θ_{k+1}. To find this new parameter estimate, the algorithm calculates a search direction and a step size. The calculation depends on whether the Hessian matrix, evaluated at θ_k, is positive definite or not.

If the Hessian matrix, $h_f(\theta_k)$, is positive definite, one can use the Newton search direction defined by

$$s_k = -h_f(\theta_k)^{-1}\, g_f(\theta_k) \tag{1}$$

The new parameter vector is then calculated as

$$\theta_{k+1} = \theta_k + \lambda_k\, s_k \tag{2}$$

where λ_k is a scalar, called the *step size*. This step size is chosen as the largest value of the series 2^{-i} $(i = 0, 1, 2, \ldots)$ so that the first-order Armijo condition (cf. McCormick [1983, p. 134]) is fulfilled:

$$f(\theta_{k+1}) - f(\theta_k) \le \mu\, \lambda_k\, s_k'\, g_f(\theta_k) \qquad 0 < \mu < 1 \tag{3}$$

The scalar constant μ is given a default value of 0.2. In most cases it is not necessary to try other values. This can be done, however, by using the `amue` option.

Two more parameters can be used to control the line search. The default maximum number, 50, of sub-iterations during line search can be changed with the `mxitl` option; and the minimum step length, default 10^{-10}, can be changed with the `smin` option.

[1] This has been proposed, for instance, by Polak [1971, p. 304].

If the Hessian matrix at θ_k is not positive definite the Newton search direction is not available. This is possible in intermediate iteration steps, in particular, if the starting values are not close to the optimizing point. Of course, it should not be the case in the final iteration steps because a local minimum requires that the Hessian matrix is positive definite.

If the algorithm finds that the Hessian matrix at θ_k is not positive definite, its further behavior depends on the type of modification. With type I (`mina=5`), the algorithm reduces then to a simple gradient method, that is the search vector is determined as the negative of the gradient, $s_k = -g_f(x_k)$. The algorithm then proceeds in the same way as explained above, i.e. the line search is based on the first-order Armijo condition.

With type II of the algorithm (`mina=6`), the search vector is build from two parts. The first part is, again, the negative of the gradient. A second part is added to reach, hopefully, a region of negative (non-positive) curvature of the function. This is tried by using the eigenvector, e_k, of the Hessian matrix that is associated with its minimum eigenvalue.[2] Because the minimum eigenvalue is less than, or equal to zero, e_k is a vector of non-positive curvature,

$$e_k' \, h_f(\theta_k) \, e_k \leq 0 \tag{4}$$

The sign of e_k is chosen so that it can be used as part of the search direction, as a vector with non-ascent direction

$$e_k' \, g_f(\theta_k) \leq 0 \tag{5}$$

The new parameter estimate is calculated as

$$\theta_{k+1} = \theta_k + \lambda_k \, s_k + \sqrt{\lambda_k} \, e_k \tag{6}$$

λ_k is again a scalar step size, now the largest of the series 2^{-i} that fulfills the second-order Armijo condition (cf. McCormick [1983, p. 135])

$$f(\theta_{k+1}) - f(\theta_k) \leq \mu \, \lambda_k \left[s_k' g_f(x_k) + \frac{1}{2} \, e_k' \, h_f(x_k) \, e_k \right] \tag{7}$$

where μ is defined the same way as with the first-order Armijo condition (3). With both modifications of the standard Newton algorithm it should be possible, not in all but in many cases, to reach within a few iteration

[2] The algorithm to calculate the minimum eigenvector of the Hessian matrix is adopted from Sparks and Todd [1973].

steps a region of negative curvature where the standard method applies. One can expect that the type II algorithm is more efficient than type I. However, the calculation of e_k is expensive, and the algorithm is slowed down in this case; therefore, the default algorithm in TDA is type I (`mina=5`).

For both versions of the Newton methods one has a choice of three convergence criteria, selectable with the `crit` parameter.

1. If `crit=1`, the algorithm uses the Euclidean norm of the gradient. If this is less than, or equal to a given tolerance value, the algorithm assumes to have reached a local minimum and terminates. The default value is 10^{-6}, but can be changed with the `tolg` option.

2. If `crit=2`, the algorithm uses the change in function values during the iterative process. The algorithm terminates if the absolute difference of the function values in two successive iterations is less than, or equal to a given tolerance. The default value, 10^{-12}, can be changed with the `tolf` option.

3. If `crit=3`, the algorithm uses changes in the function arguments to decide about convergence. First, for each of these arguments, the relative change across two successive iterations is calculated. Then, the algorithm terminates if *the maximum* of these changes is less than, or equal to a given tolerance. The default value is 10^{-4} and can be changed with the `tolp` parameter.

The default convergence criterion is a combination of the first two criteria. Convergence is assumed whenever the Euclidean norm of the gradient is less than `tolg` or the decrease of the function is less than `tolf`. If a specific criterion is selected with the `crit` option, then only this criterion is used to check convergence. All convergence checks are based on the *scaled* function.

In addition to these convergence checks, the algorithm terminates if the maximum number of iterations is reached. The default value for both Newton algorithms is 20 iterations. This can be changed with the `mxit` option. But, as a general rule, if these algorithms do not reach convergence in less than 20 iterations there is a high probability that the model is badly specified.

Another point of consideration is how to evaluate the function during step size search. This could be done without calculation of derivatives because they are not needed for the step size search. However, in many cases the Armijo conditions can be fulfilled with a unit step size, that

is with only a single sub-iteration. It is more efficient, then, to have the derivatives already calculated since they can be used for the next iteration step. Therefore, as a default, all function evaluations during the step size search include the calculation of derivatives. If this default is not optimal, it can be changed with the `sst=1` option (default is `sst=0`).

5.6.3 Protocol File

When TDA runs a minimization algorithm, basic results are written into the program's standard output and standard error output. The standard output will indicate whether the algorithm has converged and found a local optimum and also provides information about the number of iterations, function (log-likelihood) values, and parameter estimates. While the algorithm is running, some information about the iterations is also written into the standard error output. These messages will appear on the screen even if the standard output has been redirected into an output file.

In addition, one can request a more detailed protocol of the calculations performed by the selected minimization algorithm. The option to request such a protocol file is

```
prot = name_of_protocol_file,
```

TDA then creates an output file with the specified name and writes additional information into this file. By default, only an iteration protocol is written into the output file. More information about parameter values and derivatives (if any) can be requested by using `prot1` instead of `prot`. (An additional option, `prot2`, is basically only for test purposes.) The print format for the protocol file can be controlled with the

```
pfmt = print_format_for_protocol_file,
```

parameter. The default is `pfmt=-19.11`. If the `prot` parameter specifies the name of an already existing file, it is overwritten without warning. One can use `prota`, instead of `prot`, to request that the new information is appended to the the end of an already existing file.

5.6.4 Starting Values

Iterative procedures for function minimization require starting values for the parameters to be estimated. Although TDA provides default starting values, they are sometimes not good enough to find a solution. The user has then the option to provide his own starting values instead of TDA's default values. There are two options.

1. One can use the parameter

```
xp = list_of_starting_values,
```

The right-hand side must provide a sequence of numerical values separated by commas. They will be used, in the same order, as starting values for the model parameters.

2. Alternatively, one can use the parameter

```
dsv = name_of_file_containing_starting_values,
```

The file must be a free-format data file containing the starting values as first entries in each record. Given that a model has n parameters, TDA sequentially reads the first n data records and takes the first numerical entry in the ith record as starting value for the ith model parameter. If the file contains more than n data records, or if there is more than one numerical entry in each record, this is ignored. If there are less than n values, the remaining model parameters get the initial value zero.

In many applications model estimation is done sequentially, starting with simple models containing only a few parameters and then adding more parameters. It is often a good idea, to use parameter estimates from a previously estimated model as starting values for a new one. This can easily be achieved by using the option

```
ppar = name_of_an_output_file,
```

This option writes the estimated parameters (and its standard errors) into the specified output file. And the resulting output file can then be used as an input file providing the starting values for a new model estimation. Of course, if the new model contains additional parameters,

starting values for these parameters should be added to the file. The print format for the output file containing parameter estimates can be controlled with the `tfmt` option. The default is `tfmt=10.4` and is identical to the print format used for parameter estimates in TDA's standard output.

5.6.5 Covariance Matrix

Let $f(\theta)$ denote the function depending on a parameter vector, θ, that has q components. To exhibit alternative estimates of a covariance matrix, we use the following definitions.

$$J(\theta) = \left[\sum_{i=1}^{n} \frac{\partial f(y_i, \theta)}{\partial \theta_j} \frac{\partial f(y_i, \theta)}{\partial \theta_k} \right]_{j,k=1,\ldots,q}$$

$$H(\theta) = \left[\sum_{i=1}^{n} \frac{\partial^2 f(y_i, \theta)}{\partial \theta_j \, \partial \theta_k} \right]_{j,k=1,\ldots,q}$$

These definitions assume that the function is summed over n data matrix cases and y_i denotes the data for the ith case. For example, f can be a log-likelihood function.

To select a method of calculation one can use the `ccov` parameter. There are three possibilities (additional background information will be given in the section on maximum likelihood estimation).

$$\begin{aligned} J(\hat{\theta})^{-1} \quad &\text{if} \quad \text{ccov} = 1 \\ -H(\hat{\theta})^{-1} \quad &\text{if} \quad \text{ccov} = 2 \\ H(\hat{\theta})^{-1} J(\hat{\theta}) H(\hat{\theta})^{-1} \quad &\text{if} \quad \text{ccov} = 3 \end{aligned}$$

Note that the `ccov` parameter works independent of the selected minimization algorithm. Having found a solution, $\hat{\theta}$, for a local minimum, TDA uses this parameter vector for an additional function evaluation that performs the desired covariance matrix calculation.

5.6.6 Constraints

As an option, one can define linear equality constraints for the parameters of a function. If θ is the parameter vector with q components, linear equality constraints can be written as

$$R\theta = d \tag{1}$$

R is a (r, q) matrix with r the number of constraints and d is a vector of dimension r. It is required that $r < q$ since otherwise there are no degrees of freedom for a change of the function values. Also, to simplify calculations, it is required that all rows of R are linear independent, that is, R should have rank r.

The problem is to minimize the function $f(\theta)$, as defined above, subject to the constraint (1). To find a solution we follow the discussion given by McCormick [1983, chap. 13]. The basic idea is to define an appropriate subspace of the original parameter space and then to perform unconstrained function minimization in this subspace. An appropriate subspace is given as the null space of R. This is easily seen by assuming that one already has a starting vector θ_s with $R\theta_s = d$. Then all vectors fulfilling this constraint can be represented as

$$\theta_s + \theta \qquad \text{with} \qquad R\theta = 0 \tag{2}$$

If R has rank r, its null space has dimension $q-r$. One can define a $(q, q-r)$ projection matrix Q to map all vectors of the $(q-r)$-dimensional space into the null space of R, that is a matrix Q with $RQ\,\tilde{\theta} = 0$ for all vectors $\tilde{\theta}$ of dimension $(q - r)$. Consequently, the constrained minimization of $f(\theta)$ is equivalent to an unconstrained minimization of the function

$$F(\tilde{\theta}) = f(\theta_s + Q\,\tilde{\theta}) \tag{3}$$

Having found an unconstrained local minimum of F, say $\tilde{\theta}_k$, a local minimum of $f(\theta)$, subject to the constraint (1), is given by $\theta_s + Q\,\tilde{\theta}_k$.

Any of the algorithms discussed above can be used to find an unconstrained local minimum of $F(\theta)$. If one needs the gradient and the Hessian of F, these are given, respectively, by

$$g_F(\tilde{\theta}) = Q'\,g_f(\theta_s + Q\,\tilde{\theta}) \tag{4}$$

$$h_F(\tilde{\theta}) = Q'\,h_f(\theta_s + Q\,\tilde{\theta})\,Q \tag{5}$$

To find the projection matrix Q, we follow a method described by McCormick [1983, p. 265]. It is tried to find a (r, r) matrix R^D and a $(r, q-r)$ matrix R^I, where R^D consists of r linearly independent columns of R and R^I consists of the remaining columns of R.[1] The projection matrix Q can then be calculated as

$$Q = \begin{bmatrix} -\left(R^D\right)^{-1} R^I \\ I_{n-r,n-r} \end{bmatrix} \tag{6}$$

where $I_{n-r,n-r}$ denotes an $(n-r, n-r)$ identity matrix. In most cases this approach to calculating the projection matrix implies an interchange of the columns of R and, consequently, of the parameter vector. Information about the construction of Q and the implied column interchange can be found in the protocol file if requested with the `prot1` option.

If convergence has been reached in the reduced parameter space, say at $\tilde{\theta}_k$, and if it has been possible to calculate the covariance matrix in this space, the covariance matrix for the constrained model in the original space is calculated according to

$$\mathrm{Cov}[\hat{\theta}] = Q\,\mathrm{Cov}[\tilde{\theta}_k]\,Q' \tag{7}$$

This applies to all three options for covariance matrix calculation described in **5.6.5**). The covariance matrix is first calculated in the reduced parameter space and then projected (expanded) into the original parameter space.

Syntax. The syntax to define equality constraints follows formula (1). The basic key word is `con` to be used in the following way:

$$\begin{aligned} \text{con} &= r_{11}b_1 \pm \cdots \pm r_{1q}b_q = d_1, \\ \text{con} &= r_{21}b_1 \pm \cdots \pm r_{2q}b_q = d_2, \\ &\vdots \\ \text{con} &= r_{r1}b_1 \pm \cdots \pm r_{rq}b_q = d_r, \end{aligned} \tag{8}$$

The d_i and r_{ij} values must be given as numerical constants (floating point numbers); if zero, they can be omitted. The $b_1, b_2, \ldots, b_q$ refer to the model parameters and must be given as `b1,b2,b3,...`, independent of how the model parameters might be called in the model description.

[1] All calculations are done by a function adopted from the routine `LHHFTI` published by Lawson and Hanson [1974].

The ordering must conform to the structure of the model's parameter vector; `b1` refers to the first parameter, `b2` refers to the second parameter, and so on.

Note that using constraints might conflict with user-supplied starting values. If there is no solution of the constraints with the given starting values, TDA simply uses one of the feasible solutions as new starting values. A message is then given in the standard output.

Example 1 To illustrate, we add the constraint $x = -y$ to the function minimization command already used in Example 5.6.1-1. The new command (see command file `fmin3.cf`) is

```
fmin( con = b1 + b2 = 0 ) = (x - 5)^2 + (y + 6)^2;
```

The solution is $x = 5.5, y = -5.5$.

5.6.7 Possible Difficulties

There is no guarantee that the algorithms described in 5.6.2 will actually find a local maximum of the function. Several problems may occur during the iterative process of function minimization, mainly as a consequence of inappropriate starting values, numerical problems like rounding errors and overflows and underflows in function evaluations, or as a consequence of an ill-conditioned Hessian matrix caused by collinearities of covariates. Concentrating on the two types of Newton algorithms, we shortly describe the problems that are recognized by TDA.

1. The Hessian matrix may not be positive definite as is requested for a local minimum. In many cases this problem may be solved by one of the modifications of the Newton algorithm described in 5.6.2. However, it is quite possible that the algorithm does not finally reach a region of negative curvature with a positive definite Hessian. In this case the algorithm fails definitely and one should check the model specification and/or try with some other starting values.

2. It is possible that the algorithm does not find a positive step size to fulfill the Armijo conditions. Sometimes this problem can be solved by defining a smaller value for the minimum step size constant, see the `smin` parameter. Normally, one has to try with other starting values, or to try with another algorithm.

3. Another problem is that the algorithm may be unable to decrease the function value so far as is required by the convergence criterion. For instance, it may not be possible to find a gradient with a norm less than some very small tolerance. In most cases this problem is a consequence of the limited accuracy of function evaluations and/or of numerical underflows and overflows. Sometimes it may also happen that the log-likelihood becomes positive. If this problem occurs one should check the required accuracy for convergence or select another convergence criterion.

4. The algorithms may fail because convergence could not be reached within the predefined maximum number of iterations. Of course, it is easily possible to increase this maximum using the `mxit` parameter. However, in most circumstances this can be taken as a serious hint that there

is a problem with the model specification and/or the data.

5. Finally, the algorithm may give rise to senseless results if there is high collinearity across some of the covariates. One should note that such problems are not detected by TDA's minimization algorithms. In most cases, collinearity shows up in other problems, caused by an ill-conditioned Hessian.

To help in diagnostics if the minimization procedure fails, one can request a protocol of the iterations with the `prot` option, see **5.6.3**. In addition, if the program encounters numerical problems, it prints some information about these problems into its standard output. The following types of numerical problems are most important:

1 Numerical underflows or overflows in the evaluation of elementary functions like exponential and logarithm.

2 Numerical underflows or overflows in the evaluation of the function and/or its derivatives.

4 If during the iteration process the Hessian matrix is not always positive definite.

5 If problems occur in evaluating type 5 variables (containing the `time` operator) while calculating the likelihood for transition rate models.

6 If problems occur in the calculation of probabilities that should have values between 0 and 1.

7 Problems in the calculation of the incomplete gamma integral.

8 Problems in the calculation of the incomplete beta integral.

9 If problems occur during calculation of the covariance matrix.

10 If numerical integration cannot achieve the required accuracy.

None of these problems implies that the algorithm has failed. However, the reliability of the results should be checked, then, by running the procedure once more with different starting values.

6. Statistical Procedures

This part contains the following sections.

6.1 Introduction to Statistical Procedures

This chapter is intended to serve as a collection of introductory remarks for the discussion of statistical procedures in subsequent sections. Currently, we have just a few remarks about missing values and case weights.

6.1.1 Missing Values

6.1.2 Case Weights

6.1.1 Missing Values

The term *missing value* is used in two different ways. One meaning refers to data generation. When the `nvar` command reads a data file, it might not be able to find the expected data required to create the values for a variable. These are then "missing values", substituted by specific numerical missing value codes as explained in **2.2**. However, when the data matrix has been created successfully, it only consists of numerical values, and TDA no longer makes any distinction between valid and "missing" values.

Consequently, there is no direct support for the second meaning of the term *missing value* that refers to the input data given to a statistical procedure. In principle, each statistical procedure should be given some information about the character of its input data, whether they are "exact", have more or less errors, or are totally missing. Currently, TDA's statistical procedures do not conform to this requirement. The user is responsible for the data that are given to the procedures. When there are missing values, they must be excluded explicitly by appropriate case selection commands.

We intend to deal with the missing value in a more satisfactory way in a future version of TDA, based on a notion of set-valued variables. This should allow to specify, for each variable, a range of possible values. And we hope that we will be able to develop at least some basic statistical procedures that can use such kind of input data.

6.1.2 Case Weights

Many procedures in TDA are able to recognize case weights. For historical reasons, there are two different approaches. Some command, as e.g. the `lsreg` command, require a local definition of case weights; some other commands require a definition of case weights with a separate command, `cwt`. The syntax of the `cwt` command is

```
cwt = W;
```

where `W` is the name of a variable which must be contained in the current data matrix. The values of `W` are then interpreted as weights for the corresponding rows of the data matrix. These values must be non-negative, and the sum of weights for all cases in the data matrix, or cases selected with the `tsel` command, must be strictly positive. If one of these conditions is not met the program will terminate with an error message. Alternatively, one can use the command as

```
cwt (wnorm=s) = W;
```

where `s` is a positive number. The weights are modified in such a way that the sum of weights for the currently selected data matrix rows becomes equal to `s`. When simply using

```
cwt (wnorm) = W;
```

the sum of weights becomes equal to the currently selected number of cases. A specification of case weights remains active until substituted by another `cwt` command, or explicitly turned off with the command

```
cwt = off;
```

6.2 Elementary Descriptive Procedures

6.2.1 Mean and Range of Variables

The `dstat` command is to request basic information about variables. The syntax is shown in the following box.

```
dstat (
      grp=...,              group variables
      fmt=...,              print format, def. 10.4
      df=...,               write to output file
      mppar=...,            create matrix
) = varlist;
```

All parameters are optional. The command uses the variables specified in `varlist` on its right-hand side or, if `varlist` is omitted, all currently defined variables.

1. One can give a list of group variables with the parameter

```
grp = G1,G2,...,
```

Each variable on the right-hand side is then taken to define a group consisting of all cases where the variable has a nonzero value, and all calculations are done separately for each group.

2. Results are written into the standard output. The print format can be controlled with the `fmt` parameter, default is `fmt=10.4`. If the name of an output file is specified with the `df` parameter, the results are also written into that output file.

3. The `mppar` parameter can be used to create a matrix that contains, in each of its rows, the calculated statistics as described below. By default, the matrix has 6 columns and the first column is zero. If there are groups, then the first column will contain the corresponding group number.

4. For each variable the command calculates the following quantities:

1. the minimum value
2. the maximum value

Box 1 Data file `ds1.dat`

```
1 1 -1 1.e-1
2 7 -1 1.e-2
3 1 -1 1.e-3
4 7  2 1.e-4
```

Box 2 Command file `ds1.cf`

```
nvar(
    dfile = ds1.dat,  # data file
    X1 = c1,
    X2 = c2,
    X3 = c3,
    X4 = c4,
);
dstat;
```

3. the mean value
4. the standard deviation
5. the total (sum of values)

These quantities are calculated for the currently selected cases in the data matrix, optionally modified with the `sel` and `grp` parameters. The command recognizes case weights if defined with the `cwt` command. The mean is calculated as

$$\mathrm{M}(x) = \Sigma_i w_i \, x_i \big/ \Sigma_i w_i$$

where w_i is the weight for case i; without weights, $w_i = 1$ for all cases. The standard deviation is calculated as

$$\mathrm{S}(x) = \sqrt{\Sigma_i w_i (x_i - \mathrm{M}(x))^2 \big/ \left((\Sigma_i w_i) - 1\right)}$$

To calculate standard deviations, TDA uses a two-pass algorithm; first the mean values are calculated, then the squared deviations from the mean are accumulated.[1]

Example 1 To illustrate the `dstat` command, we use the data file `ds1.dat` (Box 1) containing some arbitrary data. The command file,

[1] This is recommended in the literature studying numerical accuracy in algorithms for simple descriptive statistics; see, e.g., Neely [1966], Chan and Lewis [1979].

Box 3 Part of standard output from `ds1.cf`

```
> dstat
---------------------------------------------------
Descriptive statistics. Current memory: 148696 bytes.

Variable    Minimum     Maximum      Mean    Std.Dev.    Sum of values
-------------------------------------------------------------------
X1           1.0000      4.0000    2.5000      1.2910          10.0000
X2           1.0000      7.0000    4.0000      3.4641          16.0000
X3          -1.0000      2.0000   -0.2500      1.5000          -1.0000
X4           0.0001      0.1000    0.0278      0.0484           0.1111
```

`ds1.cf`, is shown in Box 2, and part of the standard output is shown in Box 3.

6.2.2 Empirical Distribution Functions

This section describes the `gdf` command that can be used to calculate marginal and joint distribution and survivor functions, based on possibly incomplete (censored) data. The syntax of the command is shown in Box 1. Most parameters are optional. Required is the name of a variable to be specified with the `yl` parameter, and the name of an output file to be given on the right-hand side.

Input data must be defined for $i = 1, \ldots, n$ units. Each unit can contribute observations for (a subset of) m dimensions. If available, y_{ij} is the observation for unit i in dimension j. In general, each observation consists of two parts

$$(y_{ij}, \delta_{ij})$$

where y_{ij} is the observed value, and δ_{ij} indicates whether the observation is not censored ($\delta_{ij} = 1$), or is right censored ($\delta_{ij} = 0$). The corresponding data matrix variables can be specified with the `yl` and `cen` parameters, respectively. The following combinations are possible.

1. Only observed values are specified with the `yl` parameter. Then all observations are assumed to be exact with corresponding values.

2. Observed values are specified with the `yl` parameter and a censoring indicator is specified with the `cen` parameter. An observations is then interpreted as exact at y_{ij} if $\delta_{ij} = 1$ and is interpreted as right censored if $\delta_{ij} = 0$.

By default, the command assumes a single dimension ($m = 1$) and $n =$ `NOC` units, where `NOC` is the number of cases in the current data matrix. The `grp` parameter can be used to specify multivariate data. The syntax is

```
grp = ID, L1,
```

where `ID` and `L1` are names of data matrix variables. Each block of data matrix rows where `ID` has identical values is interpreted as data for one unit. Any values are possible and since the data matrix is always sorted with respect to `ID` and `L1`, it is not required that blocks are contiguous. The `L1` variable must contain positive integers. The number of different

Box 1 Syntax for `gdf` command

```
gdf (
    opt=...,        method, def. 1
                    1 = marginal calculation
                    2 = joint calculation, method 1
                    3 = joint calculation, method 2
    prn=...,        output option, def. 0
                    0 = distribution functions
                    1 = survivor functions
                    2 = expected values
    yl=...,         variable name for observed values
    cen=...,        variable name for censoring information
    grp=ID,L1,      specification of dimensions
    sc=...,         offset for domain (method 1 and 2), def. 0
    n=...,          number of boxes in grid (method 1), def. 100
    mxit=...,       maximal number of iterations (method 1), def. 20
    tolf=...,       tolerance for convergence (method 1), def. 0.001
    d=...,          delta specification (method 2), def. 0.1
    fmt=...,        print format for output file, def. 10.4
    prot=...,       protocol file with diagnostic information
) = fname;
```

Box 2 Example data to illustrate `grp` parameter

ID	L1	YL	D
1	1	3	1
1	3	5	1
1	4	4	0
2	3	0	1
5	1	7	0
2	4	1	0

integers found in this variable is interpreted as the number dimensions. Again, it is not required that these numbers are contiguous. Each data matrix row provides one observation for the unit given by the `ID` variable and dimension given by the corresponding `L1` variable.

To illustrate, consider the data in Box 2. In this example there are three units providing six observations. The number of dimensions is $m = 3$. Dimensions are mapped to the values of the `L1` variable in ascending

order. Thus, dimension 1, 2, and 3 correspond, respectively, to `L1 = 1`, `L1 = 3`, and `L1 = 4`. The first unit (`ID = 1`) has observations for all three dimensions. The observation is exact in the first and second dimension, and right censored in the third dimension. Unit 2 contributes an exact observation to the second dimension and a right censored observation to the third dimension. The third unit (`ID = 5`) contributes a right censored observation to the first dimension.

Marginal Distribution Functions

If `opt = 1` (default), the command calculates marginal distributions, separately for each dimension present in the input data. Assuming that there are n_j observations for the jth dimension, the data are:

$$(y_{1j}, \delta_{1j}), \ldots, (y_{n_j j}, \delta_{n_j j})$$

Depending on `prn`, these data are used to calculate a distribution function (`prn=0`), a survivor function (`prn=1`), or expected values (`prn=2`).

Marginal EDF: Exact Observations

If the data for one dimension contain only exact observations, the command calculates a standard empirical distribution function,

$$F_j(y_{ij}) = \sum_{i=1}^{n_j} I(Y_j \leq y_{ij})$$

if `prn = 0`, or survivor function,

$$S_j(y_{ij}) = 1 - F_j(y_{ij})$$

if `prn = 1`. Here Y_j denotes the variable in the jth dimension and refers to the possible values y_{ij}, $i = 1, \ldots, n_j$. $I()$ denotes the indicator function. In this case, if `prn = 2`, the expected values equal the observed values.

Box 3 provides an illustration. Input data are given by the variable `YL`. The command

```
gdf (yl=YL) = df0;
```

creates the output file `df0`, the command

```
gdf (yl=YL,prn=1) = ds0;
```

Box 3 Illustration of calculations with exact observations

```
                df0 (prn = 0)                    ds0 (prn = 1)

 YL     D       Y            F        D       Y            S
 --     ------------------------      ------------------------
  1     1     -1.0000      0.1667     1     -1.0000      0.8333
  7     1      1.0000      0.3333     1      1.0000      0.6667
 -1     1      5.0000      0.5000     1      5.0000      0.5000
  5     1      7.0000      0.8333     1      7.0000      0.1667
  7     1      9.0000      1.0000     1      9.0000      0.0000
  9
```

creates the output file `ds0`.[1] The first column in the output files shows the dimension, then follow the value of the variable (sorted in ascending order) and the corresponding value of the distribution or survivor function. If the input data refer to more than one dimension, the output file will contain the same information separately for each dimension.

Marginal EDF: Right Censored Observations

A second situation occurs if the input data, for one dimension, contain both, exact and right censored observations. The command then uses the standard Kaplan-Meier procedure to calculate a marginal distribution, or survivor, function. Considering the jth dimension, the observations for Y_j are sorted in ascending order. If there are exact and right censored observations for the same value of the variable, exact observations come first, followed by right censored observations. The highest value of Y_j is always treated as not censored. Then, in the order from lowest to highest values, the mass of each right censored observation is distributed over all observations having higher values.

An example is given in Box 4. Input data are given by the variables `YL` and `DELTA`. The command

```
gdf (yl=YL,cen=DELTA) = df1;
```

creates the distribution function. Adding the parameter `prn = 1` creates the corresponding survivor function.

[1] The data file for this and the following examples is `gdf1.dat`. The command file is `gdf1.cf`. Both are contained in the TDA example archive.

Box 4 Marginal distributions with right censored observations

```
                df1 (prn = 0)                ds1 (prn = 1)

YL DELTA   D      Y          F         D      Y          S
--------   ----------------------     ----------------------
 3   1     1     1.0000     0.1667    1     1.0000     0.8333
 3   0     1     3.0000     0.3750    1     3.0000     0.6250
 2   0     1     4.0000     0.6875    1     4.0000     0.3125
 1   1     1     5.0000     1.0000    1     5.0000     0.0000
 4   1
 5   1
```

Box 5 Expected values based on marginal distribution

```
                de1 (prn = 2)

YL DELTA    D       Y      CEN    E
--------   --------------------------
 3   1      1     3.0000   1    3.0000
 3   0      1     3.0000   0    4.5000
 2   0      1     2.0000   0    4.1250
 1   1      1     1.0000   1    1.0000
 4   1      1     4.0000   1    4.0000
 5   1      1     5.0000   1    5.0000
```

Marginal EDF: Expected Values

If `prn = 2`, the `gdf` command calculates expected values based on the marginal distributions. If an observation is exact its expected value equals its observed value. If the observation is right censored, the command calculates

$$y^*_{ij} = E_{F_j}(Y_j \,|\, Y_j > y_{ij}) = \int_{y_{ij}}^{\infty} y \, dF_j \,/ \int_{y_{ij}}^{\infty} dF_j$$

where F_j is the Kaplan-Meier estimate of the marginal distribution function in the jth dimension.

To illustrate, consider the data in Box 5 (same as in Box 4). The command is now

```
gdf (yl=YL,cen=DELTA,prn=2) = de1;
```

The resulting output file, also shown in Box 5, contains four columns. The first column refers to the current dimension, the second column contains

the observed values, and the third column shows the censoring status of the observation. The last column contains the expected value. This will equal the observed value if the observation is not censored, otherwise the expected value as calculated from the Kaplan-Meier survivor function. Note that with this option, the input data are not sorted. Note also that the observation with highest value is always assumed to be uncensored.

Joint Distributions

We now discuss the calculation of joint distribution functions. This is done separately for each marginal pattern which is found in the input data. The word "marginal pattern" refers to a combination of dimensions,

$$(d_1, ..., d_{m_k}) \qquad \text{where} \qquad 1 \leq d_1 < d_2 < \cdots < d_{m_k} \leq m$$

m being the maximal number of dimensions as given by the input data. For ease of notation, the following discussion refers to a full marginal pattern, meaning that $m_k = m$.

Now, let n denote the number of units for the marginal pattern. We then have m observations for each unit, as follows:

$$(y_{i1}, \delta_{i1}), \ldots, (y_{im}, \delta_{im}) \qquad i = 1, \ldots, n$$

Based on these data, the command calculates marginal distributions (if `opt = 1`) or joint distributions (if `opt = 2` or `opt = 3`). Calculations depend on whether the data contain right censored observations. The contents of the resulting output file depend on the `prn` parameter. The command calculates a distribution function if `prn = 0`, a survivor function if `prn = 1`, or expected values if `prn = 2`.

Joint Distribution: Exact Observations

If all observations are exact, the command calculates a standard m-dimensional distribution function, defined as

$$F(y_1, \ldots, y_m) = \sum_{i=1}^{n} \prod_{j=1}^{m} I(Y_j \leq y_i)$$

or the corresponding survivor function

$$S(y_1, \ldots, y_m) = \sum_{i=1}^{n} \prod_{j=1}^{m} I(Y_j > y_i)$$

Box 6 Joint distribution with exact observations

```
ID  L1    YL        MP  D1 D2      F          Y1          Y2
-----------         ------------------------------------------
 1   1   1.0         1   1  2     0.2500    -1.0000      1.0000
 1   2   2.0         1   1  2     0.5000     1.0000      2.0000
 2   1  -1.0         1   1  2     0.5000     2.0000      1.0000
 2   2   1.0         1   1  2     0.7500     3.0000      1.5000
 3   1   3.0
 3   2   1.5       MP  D1 D2      S          Y1          Y2
 4   1   2.0       ------------------------------------------
 4   2   1.0         1   1  2     0.0000     3.0000      1.5000
                     1   1  2     0.2500     2.0000      1.0000
                     1   1  2     0.0000     1.0000      2.0000
                     1   1  2     0.5000    -1.0000      1.0000
```

The functions are calculated, and tabulated in the output file, for all data points in the input data.

To illustrate, consider the data shown in Box 6. There are four units, all having observations for two dimensions. The joint distribution function, shown in the upper half of the right part of the box, was calculated with the command

```
gdf (grp=ID,L1,yl=YL,opt=2,prn=0) = df3;
```

The corresponding survivor function was calculated with the command

```
gdf (grp=ID,L1,yl=YL,opt=2,prn=1) = ds3;
```

In both cases, the first column in the output file refers to the current marginal pattern, followed by the dimensions (values of `L1` variable) that define this pattern. The next m columns contain the observations, sorted in ascending or descending order. The final column contains the corresponding value of the distribution or survivor function.

If there are only exact observations, and `prn = 2`, the output file will simply contain the observations, a censoring indicator that always has value 1, and expected values that equal the observed values. In this case, the input data are not sorted.

Censored Observations, Method 1

We now consider a situation where the multivariate data contain right censored observations. Unfortunately, there is no simple generalization

of the 1-dimensional Kaplan-Meier procedure. Discussion in the literature has proposed several different approaches. The `gdf` command offers two methods. The first one (selected with `opt = 2`) can be used with observations which might be censored simultaneously in several dimensions. The second method (selected with `opt = 3`) can only be used with observations which are censored at most in one dimension. This section describes the first method (`opt = 2`).

This method uses an iterative EM-like procedure that tries to find a self-consistent estimate of the distribution function. In explaining the procedure we refer, again, to a full marginal pattern. Data are then given by

$$(y_{i1}, \delta_{i1}), \ldots, (y_{im}, \delta_{im}) \qquad i = 1, \ldots, n$$

In a first step, we calculate a domain as an m-dimensional interval

$$D = D_1 \times \cdots \times D_m$$

where

$$D_j = \,]\min_i\{y_{ij}\} - \sigma, \max_i\{y_{ij}\} + \sigma]$$

By default, the offset is $\sigma = 0$. This implies that observations which have an exact component on a left side of the domain, or a right censored component on a right side of the domain, will not be used.[2] In order to include all observations one can specify a positive offset with the `sc` parameter, see Box 1.

The iterative procedure is based on a partition of the domain into a grid of boxes. The jth dimension is partitioned into q_j intervals

$$i_j(k) = \,]\, l_j(k), u_j(k)\,] \qquad k = 1, \ldots, q_j$$

These intervals, and the corresponding boxes, are treated as open on the left side and closed on the right side. Since the number of boxes rapidly increases in higher dimensions, we require the user to specify a total number of boxes for the whole grid with the `n` parameter, default is $n = 100$. The command then tries to find an integer q such that $q^m \approx n$ and sets

$$q_1 = \ldots = q_m = q$$

[2]Corresponding values of the distribution function, or survivor function, will then be -1, and expected values will then equal the observed (possibly censored) values.

The minimum is $q = 1$, that is, the grid consists of only a single box. Let now $B_j = \{1, \ldots, q_j\}$. Then, each

$$(k_1, \ldots, k_m) \in B_1 \times \cdots \times B_m$$

refers to one box in the grid, namely

$$B(k_1, \ldots, k_m) = i_1(k_1) \times \cdots \times i_m(k_m)$$

We now define for each observation (y_{ij}, δ_{ij}) a subset

$$b_j(y_{ij}, \delta_{ij}) \subseteq B_j$$

containing pointers to those boxes (in the jth dimension) where the observation possibly has values. Explicitly, if the observation is exact, the definition is

$$b_j(y_{ij}, \delta_{ij}) = \left\{ k \in B_j \,|\, y_{ij} \in i_j(k) \right\}$$

and if the observation is right censored, the definition is

$$b_j(y_{ij}, \delta_{ij}) = \left\{ k \in B_j \,|\, \exists\, \delta > 0 : \; y_{ij} + \delta \in i_j(k) \right\}$$

Then, for each unit i,

$$\bar{b}_i = b_1(y_{i1}, \delta_{i1}) \times \cdots \times b_m(y_{im}, \delta_{im})$$

provides pointers to those boxes in the domain where unit i has, possibly, an m-dimensional value. Of course, $\bar{b}_i$ will be empty, if unit i has a component not covered by the domain.

Using these notations, Box 7 shows the iterative algorithm.[3] The maximal number of iterations can be specified with the `mxit` parameter, default is 20. Convergence is assumed if

$$\max_{k_1,\ldots,k_m} \left\{ \,|\, f(k_1, \ldots, k_m) - f'(k_1, \ldots, k_m) \,|\, \right\} \;\leq\; \text{TOLF}$$

where $f'()$ refers to the density from the previous iteration. By default, TOLF $= 0.001$; other values can be specified with the `tolf` parameter.

Information about the number of iterations and the final value of the convergence criterion is given in the standard output. In any case,

[3]The notation '$+=$' means that the expression on the right-hand side is added to the expression on the left-hand side.

Box 7 Iterative algorithm for joint distribution (method 1)

(1) $\forall\,(k_1,\ldots,k_m) \in B_1 \times \cdots \times B_m \;:\; f^*(k_1,\ldots,k_m) = 0$

(2) $\forall\, i\;\forall\,(k_1,\ldots,k_m) \in \bar{b}_i \;:\; f^*(k_1,\ldots,k_m) \mathrel{+}= \dfrac{1}{n\,|\,\bar{b}_i\,|}$

(3) $\forall\,(k_1,\ldots,k_m) \in B_1 \times \cdots \times B_m \;:\; f(k_1,\ldots,k_m) = 0$

(4) $\forall\, i\;\forall\,(k_1,\ldots,k_m) \in \bar{b}_i \;:$
$$f(k_1,\ldots,k_m) \mathrel{+}= \frac{1}{n}\frac{f^*(k_1,\ldots,k_m)}{\sum_{(l_1,\ldots,l_m)\in\bar{b}_i} f^*(l_1,\ldots,l_m)}$$

(5) end if convergence has been achieved, or the maximal number of iterations has been reached.

(6) $\forall\,(k_1,\ldots,k_m) \in B_1 \times \cdots \times B_m \;:$
$f^*(k_1,\ldots,k_m) = f(k_1,\ldots,k_m)$

(7) continue with (3)

depending on `prn`, the `gdf` command finally calculates a distribution function, a survivor function, or expected values. In order to explain the calculation, let

$$k_{ij} = \min\{\, k_j \,|\, k_j \in b_j(y_{ij}, \delta_{ij})\,\}$$

meaning that k_{ij} refers to the box where, in dimension j, observation y_{ij} begins. The distribution function is then calculated using the formula

$$F(y_{i1},\ldots,y_{im}) = \sum_{k_1=1,\ldots,k_{i1}} \cdots \sum_{k_m=1,\ldots,k_{im}} f(k_1,\ldots,k_m)$$

Correspondingly, calculation of the survivor function uses the formula

$$S(y_{i1},\ldots,y_{im}) = \sum_{k_1=k_{i1},\ldots,q_1} \cdots \sum_{k_m=k_{im},\ldots,q_m} f(k_1,\ldots,k_m)$$

If `prn = 2`, the command calculates expected values. To explain this option, let $(y_{i1},\ldots,y_{im})$ be one of the m-dimensional observations. The

Box 8 Example data set (Pruitt [1993])

ID	L1	Y	CEN
1	1	1	0
1	2	6	1
2	1	2	0
2	2	4	1
3	1	3	0
3	2	5	0
4	1	4	1
4	2	3	1
5	1	5	1
5	2	2	0
6	1	6	1
6	2	7	1
7	1	7	0
7	2	1	0
8	1	8	1
8	2	8	1

expected value, in the jth dimension, will equal y_{ij} if this component is not censored. Otherwise, it is calculated by

$$\frac{\sum_{(k_1,\ldots,k_m)\in\bar{b}_i} z_j(k_1,\ldots,k_m)\, f(k_1,\ldots,k_m)}{\sum_{(k_1,\ldots,k_m)\in\bar{b}_i} f(k_1,\ldots,k_m)}$$

$z_j(k_1,\ldots,k_m)$ is the jth component of the mean value of all exact observations falling in box $(k_1,\ldots,k_m)$ or, if the box does not contain exact observations, equals the mean of $i_j(k_j)$.

Example 1. For a first illustration we use some example data from Pruitt [1993], shown in Box 8.[4] In order to estimate a distribution function, we use the command

```
gdf(opt=2,prn=0,yl=Y,cen=CEN,grp=ID,L1,n=64,sc=0.5) = df5;
```

In this example, we have used a total number of 64 boxes and added a small offset to the domain in order to cover all observations.[5] Box 9 shows the resulting output file. Estimated expected values, calculated with the `prn=2` option are shown in Box 10.

[4] The data file is `gdf2.dat`, contained in the TDA example archive.

[5] The command file is provided as `gdf2.cf`.

Box 9 Estimated distribution function

MP	D1	D2	Y1	Y2	F
1	1	2	1.0000	6.0000	0.0000
1	1	2	2.0000	4.0000	0.0000
1	1	2	3.0000	5.0000	0.0001
1	1	2	4.0000	3.0000	0.1250
1	1	2	5.0000	2.0000	0.0000
1	1	2	6.0000	7.0000	0.6290
1	1	2	7.0000	1.0000	0.0000
1	1	2	8.0000	8.0000	1.0000

Box 10 Estimated expected values

MP	D1	D2	Y1 (obs)	Y2 (obs)	CEN	Y1 (est)	Y2 (est)
1	1	2	1.0000	6.0000	0 1	5.6197	6.0000
1	1	2	2.0000	4.0000	0 1	5.5839	4.0000
1	1	2	3.0000	5.0000	0 0	6.6101	6.9849
1	1	2	4.0000	3.0000	1 1	4.0000	3.0000
1	1	2	5.0000	2.0000	1 0	5.0000	5.1561
1	1	2	6.0000	7.0000	1 1	6.0000	7.0000
1	1	2	7.0000	1.0000	0 0	7.8492	7.1540
1	1	2	8.0000	8.0000	1 1	8.0000	8.0000

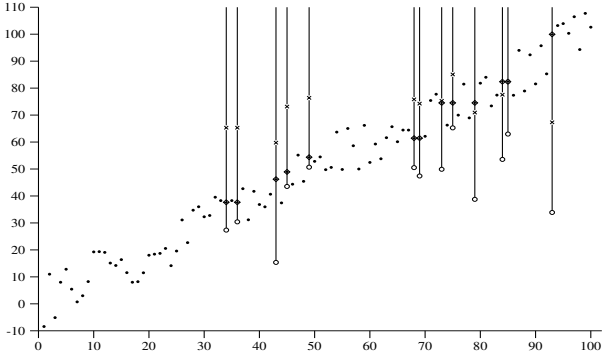


Figure 1 Illustration of observed data points and estimated expected values. Estimation with marginal Kaplan-Meier procedure (x) and with joint estimation (method 1, n = 100 boxes).

Example 2. For a second illustration, we create 100 data points

$$(y_{i1}, y_{i2}) \qquad i = 1, \ldots, 100$$

where

$$y_{i1} = i,\ y_{i2} = i + r_i$$

and r_i are random numbers which are equally distributed in $[-10, 10]$. We then have randomly censored 13 of these data points in the second dimension. The resulting data points are shown in Figure 1.

We then estimated expected values, first using a marginal Kaplan-Meier procedure.[6] The resulting estimated values are indicated in Figure 1 by cross (x) symbols. We then used the iterative procedure described above to estimate expected values. The resulting estimated values are indicated in Figure 1 by ♦ symbols. They obviously provide somewhat better estimates.

Censored Observations, Method 2

We now describe an alternative approach (selected with `opt = 3`) that can be used when the observations are censored in at most a single dimension. The basic idea is quite simple: we use a local Kaplan-Meier procedure based on all observations that are available in one of the censored dimensions.

Box 11 explains the algorithm. It is controlled by parameters Δ_j that define the size of the subsets of the domain used for the local Kaplan-Meier procedure. These parameters are calculated by using the parameter d that can be specified by the user, see Box 1. Then

$$\Delta_j = d\, W_j$$

where W_j denotes the width of the domain in dimension j. Default is $d = 0.1$.

The algorithm results in densities, $f(i)$, for all data points i that do not contain censored observations. Depending on `prn`, they are finally used to calculate a distribution function

$$F(y_{i1}, \ldots, y_{im}) = \sum_{(y_{k1}, \ldots, y_{km}) \le (y_{i1}, \ldots, y_{im})} f(k)$$

[6] The command file for data generation and estimation is provided as `gdf3.cf` in the TDA example archive.

Box 11 Algorithm for joint distribution (method 2)

(1) Sort observations $(y_{i1}, \ldots, y_{im})$ in ascending order, first wrt first component, then wrt second component, and so on; in case of ties, exact observations precede censored observations.
(2) for $i = 1, \ldots, n$: $f(i) = 1/n$
(3) $i = 1$

(4) If $(y_{i1}, \ldots, y_{im})$ is exact in all components, continue with step (8).

(5) Let $(y_{i1}, \ldots, y_{im})$ be censored in dimension j_0. Calculate an index set $B(i, j_0)$ containing indices of all data points $(y_{k1}, \ldots, y_{km})$ for which:
   a) $y_{kj} > y_{ij}$
   b) $\forall\, j \neq j_0 :\ y_{kj}$ is exact
   c) $\forall\, j \neq j_0 :\ |\, y_{kj} - y_{ij}\,| \leq \Delta_{j_0}/2$

(6) $\forall\, k \in B(i, j_0) :\ f(k) \mathrel{+}= \dfrac{f(i)}{|\, B(i, j_0)\,|}$
(7) $f(i) = 0$
(8) $i \mathrel{+}= 1$
(9) if $i \leq n$ continue with (4).

if `prn = 0`, or a survivor function

$$S(y_{i1}, \ldots, y_{im}) = \sum_{(y_{k1}, \ldots, y_{km}) > (y_{i1}, \ldots, y_{im})} f(k)$$

if `prn = 1`. These values are tabulated in the output file for all data points. The data points are not sorted.

If `prn = 2`, the command calculates expected values. For each data point $(y_{i1}, \ldots, y_{im})$, if y_{ij} is exact, this will equal the corresponding expected component. If y_{ij} is censored, the expected value is calculated

Box 12 Estimated expected values (method 1 and method 2)

Y1 (obs)	Y2 (obs)	CEN	Y1 (est)	method 1 Y2 (est)	method 2 Y2 (est)
34.0000	27.3464	1 0	34.0000	37.6603	37.3177
36.0000	30.3996	1 0	36.0000	37.6603	38.0035
43.0000	15.3593	1 0	43.0000	46.2104	46.2348
45.0000	43.5366	1 0	45.0000	48.9264	48.4735
49.0000	50.6783	1 0	49.0000	54.3749	51.9167
68.0000	50.5112	1 0	68.0000	61.4348	71.7563
69.0000	47.4082	1 0	69.0000	61.4348	71.7563
73.0000	49.9009	1 0	73.0000	74.5214	73.0277
75.0000	65.2591	1 0	75.0000	74.5214	73.7913
79.0000	38.7495	1 0	79.0000	74.5214	79.1623
84.0000	53.5625	1 0	84.0000	82.3746	83.4225
85.0000	62.9593	1 0	85.0000	82.3746	85.3742
93.0000	33.8836	1 0	93.0000	99.9475	103.4777

by

$$\hat{y}_{ij} = \frac{\sum_{k \in B(i,j)} y_{kj} f(k)}{\sum_{k \in B(i,j)} f(k)}$$

For the definition of $B(i,j)$ see Box 11. If this index set is empty, $\hat{y}_{ij}$ will equal the observed value, y_{ij}.

Example 3. For an illustration, we use the 2-dimensional data from example 2. There are 100 data points, 13 are censored in the second dimension. Since the data points are censored in only a single dimension, we can use both methods, 1 and 2. Box 12 shows the censored data points and estimated expected values for their censored component.[7] It is seen that, in this example, both methods give quite similar estimates.

Example 4. If `d = 1`, the algorithm uses all observations (in the censored dimension) and becomes identical with a standard marginal Kaplan-Meier procedure. For an illustration see command file `gdf5.cf` in the TDA example archive.

[7] The command file is `gdf4.cf`.

6.2.3 Quantiles

The syntax of the `quant` command is

```
quant (
    fmt=...,            print format, def. 7.2
    df=...,             output file
    mppar=...,          create matrix with quantiles
) = varlist;
```

The command calculates quantiles corresponding to the orders

0.1 0.2 0.25 0.3 0.4 0.5 0.6 0.7 0.75 0.8 0.9

for the variables given on its right-hand side. The print format can be controlled with the `fmt` parameter, default is `fmt=7.2`. The results are written into the standard output and, additionally, to an output file if requested with the `df` parameter.

The values of each variable, say X, are first sorted in ascending order, $x_1 \leq x_2 \leq \ldots, \leq x_n$. The pth order quantile is then calculated in the following way:

a) If $p \leq 1/(n+1)$, then: x_1

b) If $p \geq n/(n+1)$, then: x_n

c) Otherwise: $(1-(q-i))x_i + (q-i)x_{i+1}$

Where $q = p\,(n+1)$ and $i = \lfloor q \rfloor$ denotes the largest integer not greater than q.

The `mppar` parameter has syntax

```
mppar = matrix_name,
```

This creates a matrix with the specified name. The number of rows equals the number of variables. The number of columns is 11; each row contains the 11 quantiles for each variable.

6.2.4 Frequency Tables

The commands `freq`, `freq1`, and `freq2`, can be used to print frequency tables. In order to calculate these tables TDA uses an algorithm developed by Leathers [1977]. The syntax is shown in the following box.

```
freq (
     maxcat=...,     maximum number of categories, def. 1000
     fmt=...,        print format, def. 3.0
     sc=1,           contingency measures (only freq2)
     tfmt=...,       print format for contingency measures, def. 12.4
     df=...,         output file
) = varlist;
```

1. All variants of the `freq` command require a variable list on the right-hand side. The `freq` command creates a joint frequency distribution for all variables given on the right-hand side. The `freq1` command creates a separate frequency distribution for each of the variables. The `freq2` command expects exactly two variable on the right-hand side and creates a full two-dimensional table.

2. All variants of the `freq` command expect integer-valued variables. If the `cwt` command has been used to define case weights the commands create weighted frequency distributions.

3. The commands need to know in advance the maximum number of categories in the (joint) frequency table. This can be controlled with the `maxcat` parameter, default is `maxcat=1000`.

4. When using the `freq` or `freq1` command, the print format for the values of the variables can be controlled with the `fmt` parameter, default is `fmt=3.0` (must be an integer format).

5. The `sc=1` parameter can be used with the `freq2` command to request the calculation of some contingency measures, see **6.2.7**.

6. The frequency tables are written into the standard output. If an output file is specified with the `df` parameter, they are also written into

Box 1 Command file `ds2.cf`

```
nvar(
    dfile = ds1.dat,
    X1 = c1,
    X2 = c2,
    X3 = c3,
);
freq (
    df = f.out,
) = X1,X2,X3;
```

Box 2 Part of standard output from `ds2.cf`

```
> freq(df=f.out,)=X1,X2,X3
-------------------------------------------------
Frequency tables. Current memory: 144678 bytes.
Maximum number of categories: 1000

Frequency distribution for variable(s): X1,X2,X3
Number of categories: 4

Index   X1  X2  X3  Frequency   Pct   Cumulated   Pct
-------------------------------------------------------
    1    1   1  -1       1.00  25.00        1.00  25.00
    2    2   7  -1       1.00  25.00        2.00  50.00
    3    3   1  -1       1.00  25.00        3.00  75.00
    4    4   7   2       1.00  25.00        4.00 100.00
-------------------------------------------------------
Sum                      4.00 100.00
```

that output file.

Example 1 To illustrate the `freq` commands, we use again the data file `ds1.dat` (see 6.2.1). The command file is `ds2.cf`, see Box 1. The result of the `freq` command is shown in Box 2. In this example, there is a three-dimensional frequency distribution. The first column, labelled `Index`, counts the rows of the table; its last value is identical to the number of different table elements. But note that only table elements with a positive frequency are displayed. Then follow the values of the variables identifying the table elements, and the corresponding frequencies, given in absolute and percentage values; and finally the table shows the cumulated frequencies, again in absolute and percentage values. In this example, the table is also written into the output file `f.out`. Box 3

Box 3 Standard output from command `freq2=X2,X1`

```
Frequency distribution for variable(s): X2 X1
Number of categories: 4

Frequency|
Percent  |
Row Pct  |
Col Pct  |       1 |       2 |       3 |       4 |   Total
---------------------------------------------------------
       1 |    1.00 |    0.00 |    1.00 |    0.00 |    2.00
         |   25.00 |    0.00 |   25.00 |    0.00 |   50.00
         |   50.00 |    0.00 |   50.00 |    0.00 |
         |  100.00 |    0.00 |  100.00 |    0.00 |
---------------------------------------------------------
       7 |    0.00 |    1.00 |    0.00 |    1.00 |    2.00
         |    0.00 |   25.00 |    0.00 |   25.00 |   50.00
         |    0.00 |   50.00 |    0.00 |   50.00 |
         |    0.00 |  100.00 |    0.00 |  100.00 |
---------------------------------------------------------
Total         1.00      1.00      1.00      1.00      4.00
             25.00     25.00     25.00     25.00    100.00
```

illustrates the format of the frequency table if we use the `freq2`, instead of the `freq` command.

6.2.5 Aggregated Tables

For aggregating the values of a variable, and in particular for creating histograms, one can use the `atab` command with syntax shown in the following box.

```
atab (
     x=...,          definition of classes, no default
     fmt=...,        print format, def. 10.4
     s=1,            classes left open, def. s=0 (right open)
     r=1,            print only nonempty classes
     mppar=...,      write table into matrix
) = varlist;
```

`varlist` on the right-hand side must specify one, say X, or two, say X and Y, variables. Also required is the `x` parameter to specify a partition of the real line into classes for the X variable. The syntax is

`x =` $x_1, x_2, \ldots, x_n$

This defines $n+1$ classes: $(-\infty, x_1)$,$[x_1, x_2)$, ..., $[x_n, \infty)$. Another possible syntax is

`x =` $x_1(d)x_2$

defining classes $(-\infty, x_1)$,$[x_1 + id, x_1 + (i+1)d)$, ..., for $i = 0, 1, \ldots$ until x_2 is reached; an upper open class $[x_2, \infty)$ is added. By default, classes are closed on the left side and open on the right side. Using `s=1` chooses classes which are open on the left and closed on the right side.

Given such a partition of the real line, the program counts the frequency of X and calculates its mean for each of these classes. If the command provides a second variables, Y, then also the means of Y with respect the classes defined by the X variable are calculated. Weighted means will be used if case weights have been defined with the `cwt` command. Printing begins with the first non-empty class and ends with the last non-empty class. By default, the output table contains all classes in between. To suppress empty classes, one can set the `r=1` parameter.

Box 1 Example of an aggregated table

```
Aggregated table. Current memory: 148696 bytes.

X: X3         Minimum:      -1.0000  Maximum:       2.0000
Y: X4         Minimum:       0.0001  Maximum:       0.1000

     Lower       Upper  Frequency   Weighted    Mean(X)    Mean(Y)
-------------------------------------------------------------------
   -1.0000    -0.5000          3     3.0000    -1.0000     0.0370
   -0.5000     0.0000          0     0.0000     0.0000     0.0000
    0.0000     0.5000          0     0.0000     0.0000     0.0000
    0.5000     1.0000          0     0.0000     0.0000     0.0000
    1.0000     1.5000          0     0.0000     0.0000     0.0000
    1.5000     2.0000          0     0.0000     0.0000     0.0000
    2.0000     2.5000          1     1.0000     2.0000     0.0001
```

The `mppar` parameter has syntax

```
mppar = matrix_name,
```

This creates a matrix with the specified name which contains the table of aggregated values.

Example 1 To provide an illustration, we add the command

```
atab (x=-10(0.5)10) = X3,X4;
```

to command file `ds1.cf` (see 6.2.1). The result is shown in Box 1.

6.2.6 Covariance and Correlation

The `cov` and `corr` commands can be used to calculate a covariance or correlation matrix, respectively. The syntax is shown in the following box.

```
cov (
    fmt=...,        print format, def. 7.4
    df=...,         output file
    prn=...,        data format in output file, def. 0
    mpcov=...,      create matrix
) = varlist;

corr (
    fmt=...,        print format, def. 7.4
    df=...,         output file
    prn=...,        data format in output file, def. 0
    mpcov=...,      create matrix
) = varlist;

    prn = 0         lower triangle and main diagonal
    prn = 1         full square matrix
    prn = 2         column vector: lower triangle
    prn = 3         column vector: lower triangle and main diagonal
    prn = 4         column vector: full matrix
```

If no list of variables is given on the right-hand side, the commands use all currently defined variables. The lower triangle, including the main diagonal, of the resulting covariance or correlation matrix is written into the standard output. The print format can be controlled with the `fmt` parameter. Default is `fmt=7.4`. If the name of an output file is specified with the `df` parameter, the matrix is also written into that file. The data format depends on the `prn` parameter as indicated in the box above.

Calculation of the covariance matrix uses the formula

$$\mathrm{cov}(x, y) = \Sigma_i w_i (x_i - \bar{x})(y_i - \bar{y}) / (\Sigma_i w_i - 1)$$

Box 1 Standard output from `corr` command

```
Correlation matrix: corr

            X1       X2       X3       X4
-------------------------------------------
X1       1.0000
X2       0.4472   1.0000
X3       0.7746   0.5774   1.0000
X4      -0.8241  -0.5426  -0.3815   1.0000
```

with w_i optional case weights, if defined with the `cwt` command. Default is $w_i = 1$. Calculation of the correlation matrix uses the formula:[1]

$$\text{corr}(x, y) = \frac{\Sigma_i w_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\Sigma_i w_i (x_i - \bar{x})^2 \Sigma_i w_i (y_i - \bar{y})^2}}$$

The `mppar` parameter has syntax

```
mppar = matrix_name,
```

This creates a matrix with the specified name which contains the covariance, or correlation, matrix.

Example 1 To provide an illustration, we add the `corr` command, without parameters, to command file `ds1.cf` (see **6.2.1**). Box 1 shows the standard output.

[1] The correlation coefficients are actually calculated in a two-step procedure, according to algorithm P5 in Neely [1966, p. 498].

6.2.7 Contingency Measures

The `sc=1` parameter in the `freq2` command (see 6.2.4) allows to request a variety of contingency measures. Calculation is done as follows.

Assume a contingency table (f_{ij}) with r rows and c columns (when case weights are defined with the `cwt` command these are weighted frequencies) and define $r_i = \Sigma_j f_{ij}$, $c_j = \Sigma_i f_{ij}$, $n = \Sigma_{ij} f_{ij}$, and $e_{ij} = r_i c_j / n$. Calculations are only done with positive degrees of freedom: $(r-1)(c-1)$. Pearson's χ^2_P is calculated as

$$\chi^2_P = \Sigma_{ij} \frac{(f_{ij} - e_{ij})^2}{e_{ij}}$$

The likelihood ratio χ^2_{LR} is calculated as

$$\chi^2_{LR} = 2\,\Sigma_{ij} f_{ij} \log\left(\frac{f_{ij}}{e_{ij}}\right)$$

In both cases, `prob` is the corresponding value of the χ^2 distribution with $(r-1)(c-1)$ degrees of freedom. Additional coefficients are calculated as follows (we use the formulas given in SPSS [1991], CROSSTABS procedure):

$$\text{Phi} = \sqrt{\frac{\chi^2_P}{n}}$$

$$\text{Cramer's V} = \sqrt{\frac{\chi^2_P}{n(\min\{r,c\}-1)}}$$

$$\text{Contingency coefficient} = \sqrt{\frac{\chi^2_P}{\chi^2_P + n}}$$

The Lambda coefficients are calculated as

$$\lambda_{Y|X} = \frac{\Sigma_i \max_j\{f_{ij}\} - \max_j\{c_j\}}{n - \max_j\{c_j\}}$$

$$\lambda_{X|Y} = \frac{\Sigma_j \max_i\{f_{ij}\} - \max_i\{r_i\}}{n - \max_i\{r_i\}}$$

$$\lambda_{SYM} = \frac{\Sigma_i \max_j\{f_{ij}\} + \Sigma_j \max_i\{f_{ij}\} - \max_j\{c_j\} - \max_i\{r_i\}}{2n - \max_j\{c_j\} - \max_i\{r_i\}}$$

Goodman and Kruskal's τ is calculated as

$$\tau_{X|Y} = \frac{n\Sigma_{ij} f_{ij}/c_j - \Sigma_i r_i^2}{n^2 - \Sigma_i r_i^2}$$

$$\tau_{Y|X} = \frac{n\Sigma_{ij} f_{ij}/r_i - \Sigma_j c_j^2}{n^2 - \Sigma_j c_j^2}$$

The uncertainty coefficients are

$$U_{X|Y} = \frac{U_x + U_y - U_{xy}}{U_x} \quad \text{and} \quad U_{Y|X} = \frac{U_x + U_y - U_{xy}}{U_y}$$

with

$$U_x = -\Sigma_i \frac{r_i}{n} \log\left(\frac{r_i}{n}\right)$$

$$U_y = -\Sigma_j \frac{c_j}{n} \log\left(\frac{c_j}{n}\right)$$

$$U_{xy} = -\Sigma_{ij} \frac{f_{ij}}{n} \log\left(\frac{f_{ij}}{n}\right)$$

Calculation of Kendall's τ and related measures uses the following quantities:

$$D_r = n^2 - \Sigma_i r_i^2 \quad \text{and} \quad D_c = n^2 - \Sigma_j c_j^2$$

$$P = \Sigma_{ij} f_{ij} \left(\Sigma_{h<i}\Sigma_{k<j} f_{hk} + \Sigma_{h>i}\Sigma_{k>j} f_{hk}\right)$$

$$Q = \Sigma_{ij} f_{ij} \left(\Sigma_{h<i}\Sigma_{k>j} f_{hk} + \Sigma_{h>i}\Sigma_{k<j} f_{hk}\right)$$

Then

$$\text{Kendall's } \tau_b = \frac{P - Q}{\sqrt{D_r D_c}}$$

$$\text{Kendall's } \tau_c = \frac{\min\{r, c\}(P - Q)}{n^2(\min\{r, c\} - 1)}$$

$$\text{Gamma} = \frac{P - Q}{P + Q}$$

$$\text{Somers' } D_{X|Y} = \frac{P - Q}{\sqrt{D_c}} \quad \text{and} \quad D_{Y|X} = \frac{P - Q}{\sqrt{D_r}}$$

Box 1 Illustration of contingency measures

```
Contingency measures for table: X (X2) x Y(X1)

Number of cells with less than 5 elements:   8 (100.0 %)
Cells with expected frequency less than 5:   8 (100.0 %)
Degrees of freedom: 3

Chi-square (Pearson)                         4.0000  prob: 0.2615
Chi-square (likelihood ratio)                5.5452  prob: 0.1360

Phi                                          1.0000
Cramer's V                                   1.0000
Contingency coefficient                      0.7071
Lambda (X dependent)                         1.0000
Lambda (Y dependent)                         0.3333
Lambda (symmetric)                           0.6000
Goodman/Kruskal Tau (X dependent)            1.0000
Goodman/Kruskal Tau (Y dependent)            0.3333
Uncertainty coeff (X dependent)              1.0000
Uncertainty coeff (Y dependent)              0.5000
Kendall's tau-b                              0.4082
Kendall's tau-c                              0.5000
Gamma                                        0.5000
Somers' D (X dependent)                      0.3333
Somers' D (Y dependent)                      0.5000
```

Example 1 To illustrate, we use the command

```
freq2 (sc=1) = X2,X1;
```

with data file `ds1.dat` (see **6.2.1**), the command file is `ds2.cf`. Part of the standard output is shown in Box 1.

6.2.8 Scatterplots

This section describes the `scplot` command that can be used to create scatterplots, optionally with a regression curve. The syntax is shown in the following box.

```
scplot (
     opt=...,      type of plot, def. 1
                   1 = only symbols
                   2 = sunflower plot
                   3 = lowess regression
     s=...,        symbol type for opt=1, def. 0
     fs=...,       symbol size for opt=1, def. 2 (mm)
     nn=...,       grid definition for opt=2, def. nn=1,1
     fss=...,      symbol size for opt=2, def. 2 (mm)
     lt=...,       line type for regression lines, def. 1
     lw=...,       line width for regression lines, def. 0.2 (mm)
     sig=...,      smoothing factor for opt=3, def. 0.5
     d=...,        smoothing factor for opt=3, def. 0.5
     ns=...,       smoothing factor for opt=3, def. 0.5
     df=...,       output from lowess regression
     fmt=...,      print format for df option, def. 10.4
     sel=...,      case select option
     nc=...,       no clipping if nc=1, def. 0
) = X,Y;
```

The command requires that a PostScript plot environment is available, see 4.1. The commands also expects two variable names on the right-hand side asumed to contain the x and y coordinates of the data points, respectively. All other parameters are optional. By default, the command uses all cases in the currently active data matrix. A subset of cases can be selected with the

```
sel = expression,
```

Then only those data matrix rows are used where `expression` has a nonzero value. Also by default, the command will only plot inside the

Box 1 Command file `scplot1.cf`

```
psfile = scplot1.ps;  output file
psetup(
    pxlen =  90,      # length of x axis in mm
    pylen =  50,      # length of y axis in mm
    pxa   =  0,3,     # user coordinates on x axis
    pya   =  -1,2.5,  # user coordinates on y axis
);
plxa (sc=1,ic=10);      plot x axis
plya (sc=1,ic=0);       plot y axis

nvar(                 # create some random data
    noc = 200,
    X = rd(0,3),
    Y = sin(X) + rd,
);
scplot(               # using default opt = 1
    s = 5,            # symbol type
    fs = 1,           # symbol size
) = X,Y;
```

bounding box of the current coordinate system. The `nc=1` parameter can be used to allow for plots outside this box.

1. If `opt=1` (default) the command plots a separate symbol for each data point. This requires that a valid symbol is selected with the `s` parameter. For available symbols see 4.4.2. The symbol size can be modified with the `fs` parameter.

Command file `scplot1.cf` (Box 1) illustrates this option with some random data. The `s` parameter selects the symbol type and `fs=1` sets the symbol size to 1 mm. Figure shows the resulting plot.

2. If the number of data points is large one sometimes gets a better view of their density by using sunflower plots, see, e.g., Chambers et al. [1983, p. 107], Schnell [1994, p. 94]. This option is selected with the `opt=2` parameter. In addition, one can use the `nn` and `fss` parameters. The parameter

`nn` = n_x, n_y ,

specifies the grid. The x axis is partitioned into n_x intervals, the y axis into n_y intervals. The number of data points contained in each of the resulting cells is represented by a single sunflower symbol that shows the number of data points. The default size is 2 mm. The `fss` parameter can

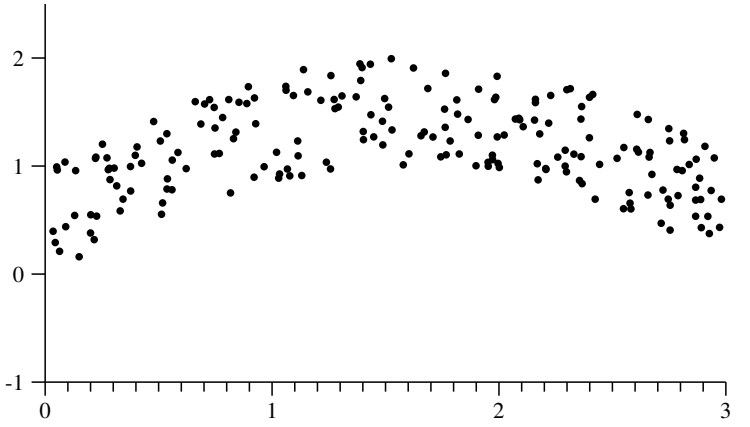


Figure 1 PostScript plot `scplot1.ps`, created with command file `scplot1.cf` shown in Box 1.

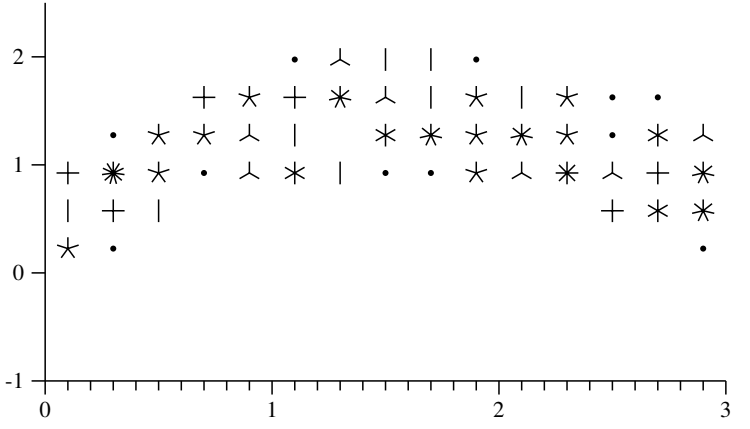


Figure 2 PostScript plot `scplot2.ps` (sunflower plot), created with command file `scplot2.cf`.

be used to specify another size.

To illustrate this option we use command file `scplot2.cf`, basically identical with `scplot1.cf`, but the `scplot` command is now

```
scplot(opt=2, nn=15,10, fss=3) = X,Y;
```

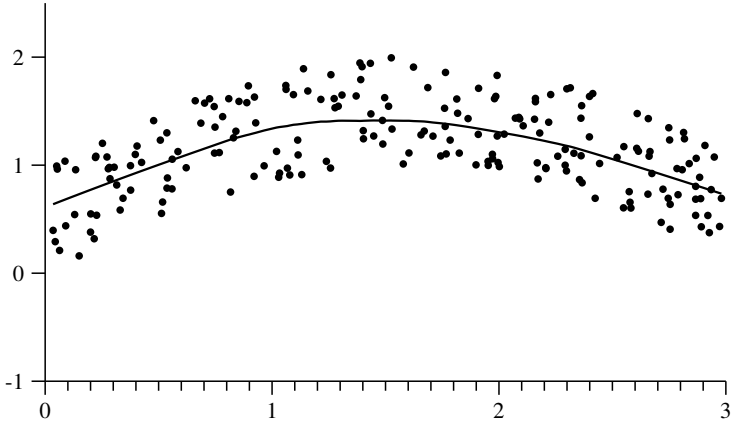


Figure 3 PostScript plot `scplot3.ps` (LOWESS regression curve) created with command file `scplot3.cf`.

Figure 2 shows the resulting plot.

3. The `opt=3` parameter can be used to add a regression curve to the scatterplot. The regression curve is calculated with the LOWESS algorithm proposed by W. S. Cleveland, a procedure for robust locally weighted regression. For an introduction see Chambers et al. [1983, p. 121]. TDA uses the algorithm developed by W. S. Cleveland (distributed via the Internet NETLIB). The algorithm can be controlled with three parameters.

1. σ, specified with `sig=`σ, should be a number between 0 and 1 and determines the fraction of data points that is used to compute each fitted value. As σ increases also the amount of smoothing increases, and computation will take more time. By default, $\sigma = 0.5$.

2. n_s, specified with `ns=`n_s, determines the number of iterations in the robust fit. If $n_s = 0$ (default) there will be no iterations and the algorithm calculates a "non-robust" regression curve.

3. δ, specified with `d =` δ, can be used to save computations. If $\delta = 0$ (default) the algorithm performs complete computations. If $\delta > 0$ it only uses data points that are spaced (about) δ apart. Remaining values are then calculated by linear interpolation.

To illustrate we use command file `scplot3.cf`. Again, basically identical

with `scplot1.cf`, but the `scplot` command is now

```
scplot(opt=3, lw=0.3, s=5, fs=1) = X,Y;
```

`opt=3` selects the LOWESS procedure with default parameters. `lw=0.3` sets the line width to 0.3 mm. We have added the `s=5` and `fs=1` parameters to include a standard scatterplot. Otherwise, the command would only plot the regression curve. (Of course, one can use the `scplot` command several times, with different options, for the same PostScript output file; and also add other plot objects.) Figure 3 shows the resulting plot.

6.2.9 Histogram Plots

This section describes the `ploth` command that can be used to create and plot histograms. The syntax is shown in the following box.

```
ploth (
     x=...,              intervals (required)
     w=...,              variable for weights
     dscal=...,          scaling factor, def. 1.0
     s=...,              type of intervals, def. 0
                         0 : intervals are left closed
                         0 : intervals are right closed
     ns=...,             option for vertical lines, def. 0
                         0 : plot vertical lines
                         0 : don't plot vertical lines
     lt=...,             line type, def. 1
     lw=...,             line width, def. 0.2 (mm)
     gs=...,             grey scale value, def. 1 (white)
     nc=...,             no clipping if nc=1, def. 0
     df=...,             print histogram data to output file
     fmt=...,            print format for df option, def. 10.4
) = name_of_a_variable;
```

The command requires a valid variable name on the right-hand side and the specification of intervals with the `x` parameter which has syntax:

```
x = x1,x2,...,xn,   or   x = x1(d)xn,
```

It is required that `x1` is not greater than the smallest value of the variable and `xn` is not smaller than the greatest value of the variable. All other parameters are optional. If the `df` option is used, the command writes the following values into the output file:

1. Begin of interval
2. End of interval
3. Number of cases in interval

Box 1 Command file `dh1.cf`

```
nvar(                                   dh1.dat
    dfile = dh1.dat,                    -------
    X = c1,                                1
);                                         3
psfile = dh1.ps;                           4
psetup(                                    5
    pxa = 0,8,                             4
    pya = 0,0.4,                           7
);                                         1
plxa(sc=1);
plya(sc=0.1);
ploth(
    x = 0,2,4,6,8,
    df = df,
) = X;
```

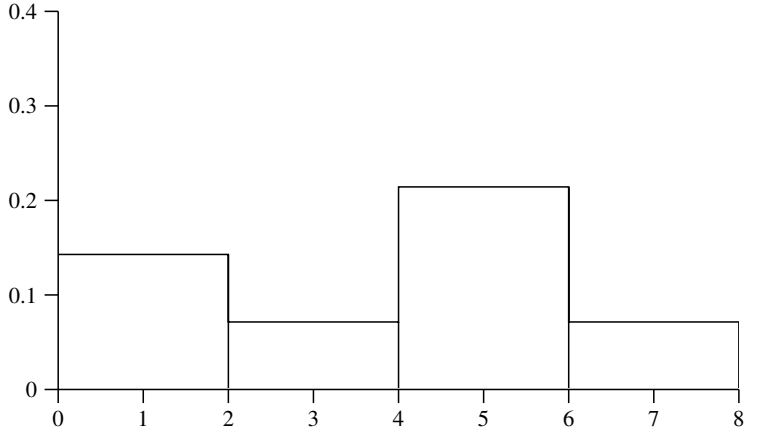


Figure 1 PostScript plot `dh1.ps`, created with command file `dh1.cf` shown in Box 1.

4. Frequency of these cases
5. Height of histogram box

As an illustration, the command file `dh1.cf`, shown in Box 1, creates the plot shown in Figure 1.

6.2.10 Density Plots

This section describes the `plotd` command that can be used to plot a density function. The syntax is shown in the following box.

```
plotd(
    x=...,          sequence of evaluation points (required)
    d=...,          bandwidth, def. 1.0
    k=...,          selection of kernel, def. 1
                    1 : uniform
                    2 : triangle
                    3 : quartic
                    4 : Epanechnikov
    dscal=...,      scaling factor, def. 1.0
    lt=...,         line type, def. 1
    lw=...,         line width, def. 0.2 (mm)
    gs=...,         grey scale value, def. 1 (white)
    nc=...,         no clipping if nc=1, def. 0
    df=...,         print histogram data to output file
    fmt=...,        print format for df option, def. 10.4
) = name_of_a_variable;
```

The command requires a valid variable name on the right-hand side and the specification of a sequence of evaluation points with the `x` parameter which has syntax:

```
x = x1,x2,...,xn,   or   x = x1(d)xn,
```

For each point x specified with the `x` parameter the command calculates

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{m} K_d(x, y_i)$$

Here it is assumed that $y_1, \ldots, y_m$ are the values of the variable specified on the right-hand side of the command. A curve connecting the values of $\hat{f}(x)$ is plotted in the current coordinate system. If the `df` option is used the x and $\hat{f}(x)$ values are also written into the specified output file.

Box 1 Command file `dh2.cf`

```
nvar(                                   dh1.dat
    dfile = dh1.dat,                    -------
    X = c1,                                1
);                                         3
psfile = dh2.ps;                           4
psetup(                                    5
    pxa = 0,8,                             4
    pya = 0,0.6,                           7
);                                         1
plxa(sc=1);
plya(sc=0.1);
plotd(
    k = 2,
    x = 0(0.1)8,
    df = df,
) = X;
```

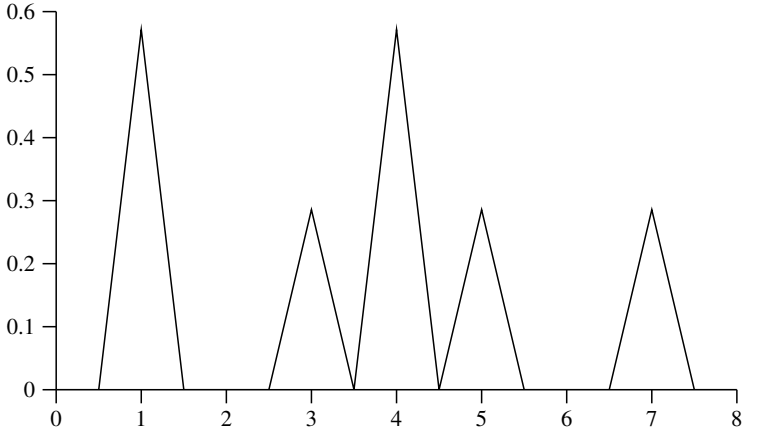


Figure 1 PostScript plot `dh2.ps`, created with command file `dh2.cf` shown in Box 1.

The user has a choice between four kernels. For a description of these kernels see **6.10.2**.

As an illustration, the command file `dh2.cf`, shown in Box 1, creates the plot shown in Figure 1.

6.4 Concentration and Inequality

This chapter is intended to describe commands that can be used to calculate concentration and inequality measures. Currently, we have only two sections.

6.4.2 Inequality Measures

In order to calculate some inequality measures one can use the `ineq` command with syntax shown in the following box.

```
ineq(
     fmt=...,       print format, def. 10.4
     df=...,        output file
     mppar=...,     write parameter into matrix
) = varlist;
```

The command requires a list of variables on the right-hand side. The other parameters are optional. The command creates a table, one line for each variable specified in `varlist`, containing the following entries:

1. Name of the variable.
2. Number of valid cases. Note that the command uses only nonnegative values of a variable; negative values are always regarded as missing values.
3. Minimum value.
4. Maximum value.
5. Mean value:

   $$\mathrm{M}(x) = \Sigma_i w_i x_i / w \quad \text{with} \quad w = \Sigma_i w_i$$

   x_i are the valid (nonnegative) values of the variable; w_i are case weights which can be provided with the `cwt` command; otherwise all $w_i = 1$.
6. Standard deviation:

   $$\mathrm{S}(x) = \sqrt{\Sigma_i w_i (x_i - \mathrm{M}(x))^2 \,/\, ((\Sigma_i w_i) - 1)}$$

7. The variation coefficient, defined as $\mathrm{V}(x) = \mathrm{S}(x)/\mathrm{M}(x)$.

Box 1 Example of inequality measures

```
Inequality measures. Current memory: 148696 bytes.

Variable Cases  Minimum   Maximum     Mean   StdDev   VCoeff     Gini
-------------------------------------------------------------------
X1           4   1.0000    4.0000   2.5000   1.2910   0.5164   0.3333
X2           4   1.0000    7.0000   4.0000   3.4641   0.8660   0.5000
X3           1   2.0000    2.0000   2.0000   0.0000   0.0000   0.0000
X4           4   0.0001    0.1000   0.0278   0.0484   1.7410   0.9262
```

8. The Gini coefficient,[1] calculated as

$$\mathrm{G}(x) = \frac{1}{2\,\mathrm{M}(x)} \frac{\Sigma_i \Sigma_{j<i} w_i w_j \,|\, x_i - x_j \,|}{\Sigma_i \Sigma_{j<i} w_i w_j}$$

The `fmt` parameter can be used to change the print format for the table entries; default is `fmt=10.4`. If the name of an output file is provided on the right-hand side of the command, a copy of the table (without header) is written into that file.

The `mppar` parameter has syntax

```
mppar = matrix_name,
```

This creates a matrix with the specified name. The number of rows equals the number of variables. Each row contains the quantities for the corresponding variable.

Example 1 To provide an illustration, we add the command

```
ineq = X1,,X4;
```

to command file `ds1.cf` (see 6.2.1). The result is shown in Box 1.

[1] See, for instance, Kendall and Stuart [1977, p. 48].

6.4.3 Segregation Indices

The `segr` command can be used to request the calculation of segregation indices. For a discussion of such indices see, e.g., James and Taeuber [1985], Deutsch et al. [1994]. The syntax of the command is shown in the following box.

```
segr (
     g=...,          name of variable defining groups
     fmt=...,        print format, def. 10.4
     df=...,         output file
     ptab=...,       additional output file
) = varlist;
```

To explain this command, assume a sample of n individuals (cases) in the current data matrix, and a variable G providing information about membership of the individuals in one of two groups:

$$G_i = \begin{cases} = 0 & \text{if individual } i \text{ is in group 0} \\ > 0 & \text{if individual } i \text{ is in group 1} \\ < 0 & \text{if no information (missing values)} \end{cases}$$

This variable G must be provided with the `g` parameter in the `segr` command. Further assume a set of classes (for instance, occupations), and each individual belonging to one of these classes, indicated by an integer variable X: individual i belongs to class j if $X_i = j \geq 0$ (otherwise there is a missing value). One or more of these X variables must be specified on the right-hand side of the `segr` command.

Given these variables, the `segr` command creates a table, one line for each X variable, containing the following entries:

1. Name of the X variable.
2. Number of classes, that is, the number of different nonnegative integer values in variable X.
3. Number of valid cases: N = number of cases with nonnegative values in G and X.

Box 1 Data file `ds2.dat`

```
G X1 X2 X3 X4
-------------
0 10  1  0  1
1  0  2  0  1
0 10  3  1  1
1  0  4  1  2
0 10  5  2  2
1  0  6 -1  2
1  0  7  2  2
```

4. Number of cases in group 0: F = number of valid cases with $G_i = 0$.
5. Number of cases in group 1: M = number of valid cases with $G_i > 0$.
6. The dissimilarity index, calculated as

$$\mathrm{D} = 0.5\,\Sigma_j \left| \frac{F_j}{F} - \frac{M_j}{M} \right| = 0.5\,\frac{\Sigma_j N_j \,|\, F_j/N_j - F/M \,|}{FM/N}$$

   where F_j = number of valid cases belonging to group 0 and class j, M_j = number of valid cases belonging to group 1 and class j, and $N_j = F_j + M_j$.

7. The variance ratio (James and Taeuber 1985, p. 6), calculated as

$$\mathrm{VR} = \frac{\Sigma_j N_j (F_j/N_j - F/N)^2}{FM/N} = \frac{\Sigma_j N_j (M_j/N_j - M/N)^2}{FM/N}$$

8. The Gini index (James and Taeuber [1985, p. 5], Deutsch et al. [1994, p. 134]), calculated as

$$\mathrm{G} = \frac{0.5}{FM}\,\Sigma_i \Sigma_j N_i N_j \left| \frac{F_i}{N_i} - \frac{F_j}{N_j} \right|$$

The table containing these entries is always written into the standard output. If the name of an output file is given with the `df` parameter, a copy of the table is written into that file. The `fmt` parameter can be used to change the print format, default is `fmt=10.4`.

In addition, one can specify the name of a second output file with the `ptab` parameter. For each X variable, it will contain a separate table with the following entries:

Box 2 Command file `ds6.cf`

```
nvar(
    dfile = ds2.dat,
    G  = c1,
    X1 = c2,
    X2 = c3,
    X3 = c4,
    X4 = c5,
);
segr(
    g=G,
    ptab = tab,
) = X1,,X4;
```

Box 3 Part of output from command file `ds6.cf`

```
Segregation measures. Current memory: 152713 bytes.
Grouping variable: G

Variable  Classes  Cases  Group_0  Group_1  D-Index  V-Ratio     Gini
-------------------------------------------------------------------------
X1              2   7.00     3.00     4.00   1.0000   1.0000   1.0000
X2              7   7.00     3.00     4.00   1.0000   1.0000   1.0000
X3              3   6.00     3.00     3.00   0.0000   0.0000   0.0000
X4              2   7.00     3.00     4.00   0.4167   0.1736   0.4167
```

1. Number of classes, based on the current X variable.
2. Number of valid cases.
3. Number of cases in group 0.
4. Number of cases in group 1.
5. Contribution of the class to the dissimilarity index, that is, $|\, F_j/F - M_j/M \,|$.

Example 1 To provide an illustration, we use data file `ds2.dat`, shown in Box 1, and the command file `ds6.cf` (Box 2). Part of the standard output is shown in Box 3.

6.5 Describing Episode Data

This chapter discusses some elementary descriptive procedures for episode data. They require a valid episode data structure, see **3.3.2**.

6.5.1 Life Tables

The life table method allows to calculate nonparametric estimates of the survivor function, the density function, and transition rates, for durations given in a set of episodes. (An extensive discussion of this method has been given by Namboodiri and Suchindran [1987]; a compilation of all commonly used formulas for the single transition case may be found in Smith et al. [1970].) There are two drawbacks because it is necessary to group the durations according to some intervals on the time axis. First, results depend more or less on arbitrarily defined time intervals; and second, the method should only be used if there is a relatively large number of episodes so that estimates conditional for each interval are reliable. Therefore, in most applications one normally prefers the Kaplan-Meier procedure discussed in **6.5.2**.

Given a valid definition of single episode data as explained in **3.3.2**, one can estimate life tables with the `ltb` command. The syntax is explained in Box 1. The command needs the specification of an output file on the right-hand side. The life tables are then written into this output file. The print format can be specified with the `fmt` parameter; default is `fmt=7.5`.

Time Periods. Life table estimation, like some other methods for episode data, requires a specification of time periods (intervals) for the process time axis that was used to define the episode data. This is done by defining split points on the time axis

$$0 \leq \tau_1 < \tau_2 < \tau_3 < \ldots < \tau_q$$

With the convention that $\tau_{q+1} = \infty$, there are q time intervals each including the left limit, but not the right one.

$$I_l = \{t \mid \tau_l \leq t < \tau_{l+1}\} \qquad l = 1, \ldots, q$$

Given these time intervals, calculation of life tables by TDA is always done using episode durations. (This implies that the method cannot be used with split episode data.) In the following discussion we will therefore assume that all episodes have equal starting times at the origin, zero, of a common time axis. In addition we will assume that the time intervals start at zero, i.e. $\tau_1 = 0$.

Box 1 Syntax for `ltb` command

```
ltb (
     tp=...,        time periods, no default
     cfrac=...,     fraction of censored cases that are counted
                    as part of the risk set, def. 0.5
     grp=...,       group indicator variables, def. without groups
     fmt=...,       print format for output file, def. fmt=7.5
     ap=1,          append data to end of output file
) = name_of_output_file;
```

For the definition of time periods one must use the `tp` parameter in the `ltb` command. The syntax is

```
tp = t1,t2,t3,...,tq,
```

where `t1,t2,t3,...,tq` are non-negative numbers in a strictly ascending order. If the first time point, `t1`, is greater than zero, TDA inserts an additional interval beginning at zero and ending at `t1`. Alternatively, one can use the syntax

```
tp = t1 (d) t2,
```

The expression is then expanded to provide the sequence `t1, t1 + d, t1 + 2d,...`, until `t2`. It is possible to combine both ways of defining time periods.

Given a valid definition of time periods, the calculation of life tables depends on the input data.

1. If there are case weights defined with the `cwt` command, these weights are used with all calculations. In particular with large data sets where durations are heavily tied this is an useful option.

2. If the input data are partitioned into groups (see below), life table estimation is done separately for the episodes in each group.

3. If there is more than one origin state, life table calculation is done separately for each subset of episodes having the same origin state. Consequently, life table calculation is always conditional on a given origin state.

4. If for a given origin state there is only a single destination state, an ordinary life table is calculated; if there are two or more destination states a so-called multiple-decrement life table is calculated.

It should be stressed that TDA always uses durations, calculated as the difference between starting and ending times. Consequently, episode data that have been split with the method of episode splitting cannot be used as input for life table estimation. If one or more starting times in the input data are not zero, TDA gives a warning message that the results will be probably wrong.

Grouping Episodes. It is possible to partition a set of episodes into groups. This can be done with the optional `grp` parameter in the `ltb` command. The syntax is

```
grp = G1,G2,...,
```

where `G1,G2,...` are names of variables. These variables are then interpreted as indicator variables defining groups. Each of the variables defines a separate group consisting of all episodes (data matrix rows) where the variable has a value not equal to zero.

Single Transitions. All formulas used in the calculation of single transition life tables are based on the quantities $I_l (l = 1, \ldots, q)$.

E_l = the number of episodes with events in I_l

Z_l = the number of censored episodes ending in I_l

defined for the intervals (time periods) $I_l (l = 1, \ldots, q)$. The next important point is the definition of a *risk set*, $\mathcal{R}_l$, for each of the time intervals, that is the set of units (episodes) that are at risk of having an event during the lth interval.[1] To take into account episodes that are censored during the interval this is done in two steps. First the number of episodes, N_l, that enter the lth interval, is defined recursively by

$$N_1 = N, \quad N_l = N_{l-1} - E_{l-1} - Z_{l-1}$$

In a second step one has to decide how many of the episodes that are censored during an interval should be contained in the risk set for that interval. We shall assume a constant ω $(0 \leq \omega \leq 1)$ for the definition

[1] We generally denote the risk set by the symbol $\mathcal{R}$, the number of units contained in the risk set by the symbol R.

of the fraction of censored episodes that should be contained in the risk set. The default value is $\omega = 0.5$, but can be changed with the `cfrac` parameter.[2]

The number of elements in the risk set is defined, then, by

$$R_l = N_l - \omega\, Z_l$$

Using these basic quantities, it is easy to define all other concepts used in the life table setup. First the conditional probabilities of having an event in the lth interval, q_l, and for surviving the interval, p_l, are

$$q_l = \frac{E_l}{R_l} \quad \text{and} \quad p_l = 1 - q_l$$

As an implication, one gets the following estimator for the survivor function[3]

$$G_1 = 1, \quad G_l = p_{l-1} G_{l-1}$$

Having estimates of the survivor function, the density function is evaluated approximately at the midpoints of the intervals as the first derivative

$$f_l = \frac{G_l - G_{l+1}}{\tau_{l+1} - \tau_l} \qquad l = 1, \ldots, q-1$$

Of course, if the last interval is open on the right side, it is not possible to calculate the survivor function for this interval. Also, estimates of the transition rate, r_l, are calculated at the midpoints of the intervals. They are defined by

$$r_l = \frac{f_l}{\bar{G}_l} \qquad \text{where} \qquad \bar{G}_l = \frac{G_l + G_{l+1}}{2}$$

and this can also be written as

$$r_l = \frac{1}{\tau_{l+1} - \tau_l} \frac{q_l}{1 - q_l/2} = \frac{1}{\tau_{l+1} - \tau_l} \frac{E_l}{R_l - E_l/2}$$

[2] Cf. the discussion given by Namboodiri and Suchindran [1987, p. 58].

[3] Note that the survivor function is calculated at the begin of each interval. Most programs use this convention in the printout of life tables. An exception is SPSS; in the life table output of SPSS the survivor function is given at the end of each interval.

Box 2 Command file `ltb1.cf` for life table estimation

```
... reading input data file rrdat.1
... set up episode data with edef command

ltb (
    tp = 0 (30) 500,    # time periods: 0,30,60,...,300
) = ltb.1;
```

Finally, it is possible to approximately calculate standard errors for the estimates of the survivor and density function, and of the transition rates, by the formulas

$$\mathrm{SE}\,(G_l) = G_l \left[\sum_{i=1}^{l-1} \frac{q_i}{p_i\, R_i}\right]^{1/2}$$

$$\mathrm{SE}\,(f_l) = \frac{q_l\, G_l}{\tau_{l+1} - \tau_l} \left[\sum_{i=1}^{l-1} \frac{q_i}{p_i R_i} + \frac{p_i}{q_i R_i}\right]^{1/2}$$

$$\mathrm{SE}\,(r_l) = \frac{r_l}{\sqrt{q_l R_l}} \left[1 - \left[\frac{r_l\,(\tau_{l+1} - \tau_l)}{2}\right]^2\right]^{1/2}$$

With large samples it may be assumed that the values of the survivor, density and rate function, divided by their standard errors are approximately standard normally distributed. It is possible, then, to calculate confidence intervals.

Example 1 To illustrate life table estimation, we use the example data discussed in **3.3.3**. An example command file, `ltb1.cf`, is shown in Box 2. Its first part, reading the data file and defining single episode data, is identical with command file `ed1.cf` (see **3.3.3**). We simply add the `ltb` command for life table estimation.

TDA's standard output in response to the `ltb` command is shown in Box 3. The table has a separate line for each life table, corresponding to each set of episodes having the same origin state. In this example there is only one set of episodes all having origin state 0. The table also shows the number of episodes and an estimate of the median, calculated by linear interpolation of the life table.

The life tables are written into the output file specified in the `ltb` command. In this example, there is only a single life table. As shown in

Box 3 Part of standard output from command file `ltb1.cf`

```
Life table calculation. Current memory: 220963 bytes.
Command: ltb(tp=0(30)500,)=ltb.1

SN  Org  Group   Median  Dest.States  Episodes  Weighted
-----------------------------------------------------------
 1    0    --     47.28        1          600    600.00
-----------------------------------------------------------
1 table(s) written to: ltb.1
```

Box 4, the table is in two parts. The first part shows the time intervals and the basic quantities for each of the time intervals. The last column, with header `Prob`, shows the quantities q_l, estimates of the conditional probability of having an event in the corresponding interval.

The second part shows again the time intervals and, in addition, estimates of the survivor function, the density function and the transition rate, each with its estimated standard errors. An estimate of the median is calculated, if possible, by linear interpolation of the survivor function.

Example 2 If one defines two ore more groups of episodes, TDA calculates a separate life table for each group. An example is given in command file `ltb2.cf` (not shown) where two groups (men and women) have been defined.

Multiple-Decrement Life Tables. The concept of a life table can be extended to the case of two or more transitions starting from the same given origin state. The resulting life table is sometimes called a multiple-decrement table.

The given origin state is, say, $j \in \mathcal{O}$. The basic quantities are now: $E_{jk,l}$, the number of episodes with a transition to the destination state k in the lth time interval; $E_{j,l}$, the number of episodes with any transition in the lth interval, and $Z_{j,l}$, the number of episodes that are censored in the lth interval.

Having these basic quantities, the number of episodes entering each interval and the risk set for each interval are defined analogously to the single transition case. First, the number of episodes entering an interval may be defined again recursively by

$$N_{j,1} = N_j, \quad N_{j,l} = N_{j,l-1} - E_{j,l-1} - Z_{j,l-1}$$

with N_j the set of all episodes with origin state j. The number of elements

Box 4 Output file created by `ltb1.cf`

```
# Life table. SN 1. Origin state 0.
# Cases: 600  weighted: 600

# Start of          Number   Number   Exposed     D-State 1
# Interval Midpoint Entering Censored   to Risk Events   Prob
      0.00    15.00      600       28     586.0    223 0.38055
     30.00    45.00      349       23     337.5    113 0.33481
     60.00    75.00      213       15     205.5     51 0.24818
     90.00   105.00      147       16     139.0     25 0.17986
    120.00   135.00      106       15      98.5     24 0.24365
    150.00   165.00       67        5      64.5      9 0.13953
    180.00   195.00       53        9      48.5      4 0.08247
    210.00   225.00       40        5      37.5      3 0.08000
    240.00   255.00       32        5      29.5      0 0.00000
    270.00   285.00       27        7      23.5      2 0.08511
    300.00   315.00       18        5      15.5      2 0.12903
    330.00   345.00       11        1      10.5      2 0.19048
    360.00   375.00        8        3       6.5      0 0.00000
    390.00   405.00        5        4       3.0      0 0.00000
    420.00   435.00        1        1       0.5      0 0.00000

# Start of              Survivor       D-State 1       D-State 1
# Interval Midpoint Function  Error Density   Error    Rate   Error
      0.00    15.00 1.00000 0.00000 0.01268 0.00067 0.01567 0.00102
     30.00    45.00 0.61945 0.02006 0.00691 0.00058 0.01340 0.00124
     60.00    75.00 0.41205 0.02077 0.00341 0.00045 0.00944 0.00131
     90.00   105.00 0.30979 0.01995 0.00186 0.00036 0.00659 0.00131
    120.00   135.00 0.25407 0.01922 0.00206 0.00040 0.00925 0.00187
    150.00   165.00 0.19217 0.01822 0.00089 0.00029 0.00500 0.00166
    180.00   195.00 0.16535 0.01774 0.00045 0.00022 0.00287 0.00143
    210.00   225.00 0.15172 0.01754 0.00040 0.00023 0.00278 0.00160
    240.00   255.00 0.13958 0.01748 0.00000       ** 0.00000       **
    270.00   285.00 0.13958 0.01748 0.00040 0.00027 0.00296 0.00209
    300.00   315.00 0.12770 0.01790 0.00055 0.00037 0.00460 0.00324
    330.00   345.00 0.11122 0.01900 0.00071 0.00047 0.00702 0.00493
    360.00   375.00 0.09004 0.02045 0.00000       ** 0.00000       **
    390.00   405.00 0.09004 0.02045 0.00000       ** 0.00000       **
    420.00   435.00 0.09004 0.02045 0.00000       ** 0.00000       **
# Median duration: 47.28
```

contained in the risk set for the lth episode is defined by

$$R_{j,l} = N_{j,l} - \omega\, Z_{j,l}$$

ω has the same meaning as in the single transition case, it is the fraction of the censored episodes assumed to be not at risk for an event

in the interval. Next, one can define estimates of the basic conditional probabilities

$$q_{jk,l} = \frac{E_{jk,l}}{R_{j,l}} \qquad q_{j,l} = \sum_{k \in \mathcal{D}_j} q_{jk,l} \qquad p_{j,l} = 1 - q_{j,l}$$

$q_{jk,l}$ is an estimate for the conditional probability of a transition to destination state k in the lth episode, conditional on having no event before. $q_{j,l}$ is the conditional probability of having any event in the lth interval. And $p_{j,l}$ is the conditional probability of having no event in the lth interval.

Using these conditional probabilities, an estimate of the *overall* survivor function[4] is given by

$$G_{j,1} = 1, \quad G_{j,l} = p_{j,l-1} G_{j,l-1}$$

Obviously, it is the same definition as in the single transition case and, consequently, the overall survivor function could be estimated by collapsing all different destination states into a single one.

Following the remarks given in **3.3.1** about alternative destination states, the next step should be to get estimates for the destination-specific subdistribution and subdensity functions and for the transition rates. To do this, we can first write

$$q_{jk,l} = \frac{\tilde{F}_{jk,l+1} - \tilde{F}_{jk,l}}{G_{j,l}} \tag{1}$$

with the meaning that $q_{jk,l}$ may be regarded as an estimate of the right-hand side. This is obvious if one remembers the definition of the subdistribution functions $\tilde{F}_{jk}$, defined by

$$\tilde{F}_{jk,l} = \tilde{F}_{jk}(\tau_l) = \Pr\left(T_j \leq \tau_l, D_j = k\right)$$

It is the probability of a transition to destination state k up to the beginning of the lth interval. Then, with the assumption that $\tau_1 = 0$, it follows that $\tilde{F}_{jk,1} = 0$, and a simple manipulation of (1) gives the following estimate of the subdistribution functions

$$\tilde{F}_{jk,l} = \sum_{i=1}^{l-1} q_{jk,i}\, G_{j,i}$$

[4]See the discussion of this concept in **3.3.1**.

Estimates of the subdensity functions $\tilde{f}_{jk}$ may be derived, then, by taking an approximation to the first derivative at the midpoints of the intervals, i.e. by

$$\tilde{f}_{jk,l} = \frac{\tilde{F}_{jk,l+1} - \tilde{F}_{jk,l}}{\tau_{l+1} - \tau_l}$$

And finally, one gets estimates of the transition rates by

$$r_{jk,l} = \frac{\tilde{f}_{jk,l}}{(G_{jk,l} + G_{jk,l+1})/2}$$

All destination-specific concepts add up to accordingly defined overall concepts which are the result of an ordinary life table estimation where all different destination states are collapsed into a single state.

$$F_{j,l} = \sum_{k \in \mathcal{D}_j} \tilde{F}_{jk,l}$$

$$f_{j,l} = \sum_{k \in \mathcal{D}_j} \tilde{f}_{jk,l}$$

$$r_{j,l} = \sum_{k \in \mathcal{D}_j} r_{jk,l}$$

TDA commands to request a multiple-decrement life table are identical to the single transition case. The output file containing the life table(s) is defined with the `ltb` command, time periods are defined with the `tp` command. In fact, whenever there are two or more different transitions in the input data, TDA automatically calculates multiple-decrement life tables.

Example 3 To illustrate the calculation of a multiple-decrement life table we use our main example data with three alternative destination states as defined in **3.3.3**.[5] Our example command file, `ltb3.cf` (not shown), is almost identical to the command file `ed2.cf` that was used in **3.3.3** to create episode data with three alternative destination states. We simply add a `ltb` command to request life table estimation. TDA automatically recognizes that the input data consist of single episodes with three alternative destination states and calculates a multiple-decrement life table as described above.

[5] Many more examples of multiple-decrement life tables can be found in Namboodiri and Suchindran [1987].

6.5.2 Kaplan-Meier Procedure

Another method for nonparametric estimation of survivor functions and its derivatives is the Kaplan-Meier [1958], also called product-limit method. One of the advantages of this method, compared with the life table method, is that it is not necessary to group the episode durations according to arbitrarily defined time intervals. Instead, the product-limit method is based on the calculation of a risk set at every point in time where at least one event occurred. In this way the information contained in a set of episodes is optimally used. The only drawback of this method results from the fact that all episodes must be sorted according to their ending (and starting) times, but with efficient sorting algorithms the method can be employed with fairly large sets of episodes.

This section describes the product-limit estimation method and its implementation in TDA. The options, depending on the type of input data, are essentially the same as with the life table method.

1. If there are sample weights defined with the `cwt` command, these weights are used in all calculations.

2. If the input data are split into groups, separate product-limit estimates are calculated for each of the groups.

3. If there is more than a single origin state, one or more product-limit estimates are calculated for each subset of episodes having the same origin state.

4. If there is more than a single destination state, separate product-limit estimates are calculated for each transition found in the input data.

The basic command to request product-limit estimation is `ple`. The syntax is shown in Box 1. All parameters, except the name of an output file on the right-hand side, are optional and will be explained below.

Single Transitions. We assume a sample of N episodes, all having the same origin state and are censored or have the same destination state. If groups are defined, it is assumed that all episodes belong to the same group. For the moment we also assume that all episodes have starting time zero. (This assumption is actually not necessary for product-limit calculations with TDA. For instance, it is possible to perform product-

Box 1 Syntax for `ple` command

```
ple (
     grp=,      group indicator variables, def. without groups
     csf,       compare survivor functions.
     qo=,       request table with quantiles, no default
     qt=,       request table with quantiles, no default
     fmt=,      print format for output file, def. fmt=7.5
     ap=1,      append data to end of output file.
     prot=,     protocol for test statistic, no default
     pfmt=,     print format for protocol file, def. pfmt=-19.11
) = name_of_output_file;
```

limit estimations with a set of episode that are split into parts. Results should be exactly the same before and after splitting a set of episodes.)

The first step is to consider the points in time where at least one of the episodes ends with an event. There are, say, q such points in time.

$$\tau_1 < \tau_2 < \tau_3 < \ldots < \tau_q \tag{1}$$

The second step is to define three basic quantities, all defined for $l = 1, \ldots, q$, with the convention that $\tau_0 = 0$.

$E_l =$ the number of episodes with events at τ_l

$Z_l =$ the number of censored episodes ending in $[\tau_{l-1}, \tau_l)$

$R_l =$ the number of episodes in the risk set at τ_l, denoted $\mathcal{R}_l$, i.e. the number of episodes with starting time less than τ_l and ending time $\geq \tau_l$

Note that the implied definition of the risk set allows the handling of episodes with starting times greater than zero. Also note that the risk set at τ_l includes episodes which are censored at this point in time. It is assumed that a censored episode contains the information that there was no event up to *and including* the ending time of the episode. As is sometimes said, censoring takes place an infinitesimal amount to the right of the observed ending time.

Given these quantities, the product-limit estimator of the survivor func-

tion is defined as

$$\hat{G}(t) = \prod_{l:\tau_l<t} \left(1 - \frac{E_l}{R_l}\right) \tag{2}$$

This is a step function with steps at the points in time, τ_l. The commonly used formula to calculate estimates of standard errors for the survivor function is

$$\mathrm{SE}(\hat{G}(t)) = \hat{G}(t) \left[\sum_{l:\tau_l<t} \frac{E_l}{R_l\,(R_l - E_l)}\right]^{1/2} \tag{3}$$

In addition to survivor function estimates, the product-limit method gives a simple estimate of the cumulated transition rate.

$$\hat{H}(t) = -\log\left(\hat{G}(t)\right) \tag{4}$$

This is, again, a step function. It is especially useful for simple graphical checks of distributional assumptions about the underlying durations. Some examples are given below.

Unfortunately, unlike life table estimation, the product-limit method does not provide direct estimates of transition rates. Of course, it is possible to get estimates by numerical differentiation of $\hat{H}(t)$, but this is currently not supported by TDA.

Example 1 To illustrate product-limit estimation we replicate an example given by Lawless [1982, p. 73].[1] The data are remission times for two groups of leukemia patients, one group treated with Drug 6-MP, the other was given a placebo. The data file, `rrdat.1`, is shown in Box 2. The first column gives the remission time in weeks. The second column is the censoring status (0 if censored). The third column gives the number of episodes (case weights). The fourth column is a group indicator (0 = placebo, 1 = Drug 6-MP).

For product-limit estimation of a survivor function we use command file `ple1.cf` shown in Box 3. The `ple` command specifies two groups. The output file, `ple.1` is shown in Box 4. It contains two tables, one for each of the two groups. The first two columns are to simplify access to the tables if they are used as data files. The first column assigns a unique identification number to each table, the second column simply counts the lines.

[1] The same data have been used by many other authors, for instance Lee [1980, p. 128].

Box 2 Example data `rrdat.2`

```
 C1  C2  C3  C4      C1  C2  C3  C4
  6   1   3   0       1   1   2   1
  6   0   1   0       2   1   2   1
  7   1   1   0       3   1   1   1
  9   0   1   0       4   1   2   1
 10   1   1   0       5   1   2   1
 10   0   1   0       8   1   4   1
 11   0   1   0      11   1   2   1
 13   1   1   0      12   1   2   1
 16   1   1   0      15   1   1   1
 17   0   1   0      17   1   1   1
 19   0   1   0      22   1   1   1
 20   0   1   0      23   1   1   1
 22   1   1   0
 23   1   1   0
 25   0   1   0
 32   0   2   0
 34   0   1   0
 35   0   1   0
```

Box 3 Command file `ple1.cf` for product-limit estimation

```
nvar(
    dfile = rrdat.2,      # data file

    DUR  [2.0] = c1,      # duration;
    CEN  [1.0] = c2,      # censoring status (1 = not censored)
    WT   [1.0] = c3,      # case weight (number of individuals)
    GRP  [1.0] = c4,      # group (0 = placebo, 1 = Drug 6-MP)
    GRP1 [1.0] = GRP[0], # group 0
    GRP2 [1.0] = GRP[1], # group 1
);
cwt = WT;                 # define case weights

edef(
    org = 0,              # origin state
    des = CEN,            # destination state
    ts  = 0,              # starting time
    tf  = DUR,            # ending time
);

ple (                     # request product-limit estimation
    grp = GRP1,GRP2,      # define two groups
) = ple.1;                # output file
```

Box 4 Output file `ple.1` created by command file `ple1.cf`

```
# SN 1. Transition: 0,1 - Product-Limit Estimation

# Group: GRP1
#                    Number  Number    Exposed  Survivor    Std.     Cum.
# ID Index    Time   Events  Censored  to Risk  Function   Error     Rate
   0     0    0.00        0         0       21  1.00000 0.00000 0.00000
   0     1    6.00        3         0       21  0.85714 0.07636 0.15415
   0     2    7.00        1         1       17  0.80672 0.08694 0.21478
   0     3   10.00        1         1       15  0.75294 0.09635 0.28377
   0     4   13.00        1         2       12  0.69020 0.10681 0.37078
   0     5   16.00        1         0       11  0.62745 0.11405 0.46609
   0     6   22.00        1         3        7  0.53782 0.12823 0.62024
   0     7   23.00        1         0        6  0.44818 0.13459 0.80256
#  0     8   35.00        0         5
# Median Duration: 22.42
# Duration times limited to: 23
# Cases: 18  weighted: 21

# SN 1. Transition: 0,1 - Product-Limit Estimation

# Group: GRP2
#                    Number  Number    Exposed  Survivor    Std.     Cum.
# ID Index    Time   Events  Censored  to Risk  Function   Error     Rate
   1     0    0.00        0         0       21  1.00000 0.00000 0.00000
   1     1    1.00        2         0       21  0.90476 0.06406 0.10008
   1     2    2.00        2         0       19  0.80952 0.08569 0.21131
   1     3    3.00        1         0       17  0.76190 0.09294 0.27193
   1     4    4.00        2         0       16  0.66667 0.10287 0.40547
   1     5    5.00        2         0       14  0.57143 0.10799 0.55962
   1     6    8.00        4         0       12  0.38095 0.10597 0.96508
   1     7   11.00        2         0        8  0.28571 0.09858 1.25276
   1     8   12.00        2         0        6  0.19048 0.08569 1.65823
   1     9   15.00        1         0        4  0.14286 0.07636 1.94591
   1    10   17.00        1         0        3  0.09524 0.06406 2.35138
   1    11   22.00        1         0        2  0.04762 0.04647 3.04452
   1    12   23.00        1         0        1  0.00000      **      **
# Median Duration: 6.12
# Cases: 12  weighted: 21
```

The column labelled `Time` shows the points in time where at least one event takes place. The number of events (episodes) is given in the next column. Then follows the number of censored episodes with ending times less than the actual value of the `Time` column and greater or equal to the preceding value in the `Time` column. Given this information the risk set, printed in the fifth column, is easily calculated. The last three

Box 5 Part of output file `ple.2`

```
SN 1. Transition: 0,1 - Product-Limit Estimation

                    Number  Number    Exposed  Survivor   Std.     Cum.
ID Index    Time    Events  Censored  to Risk  Function   Error    Rate
 0     0    0.00         0         0      600   1.00000 0.00000 0.00000
 0     1    2.00         2         0      600   0.99667 0.00235 0.00334
 0     2    3.00         5         1      597   0.98832 0.00439 0.01175
 0     3    4.00         9         2      590   0.97324 0.00660 0.02712
 0     4    5.00         3         0      581   0.96822 0.00717 0.03230
 0     5    6.00        10         1      577   0.95144 0.00880 0.04978
 0     6    7.00         9         0      567   0.93634 0.00999 0.06578
 ..................................................................
 0    40   41.00         3         1      277   0.50585 0.02092 0.68152
 0    41   42.00         1         1      273   0.50399 0.02093 0.68519
 0    42   43.00         2         0      272   0.50029 0.02094 0.69257
 0    43   44.00         5         1      269   0.49099 0.02096 0.71133
 0    44   45.00         1         1      263   0.48912 0.02096 0.71514
 0    45   46.00         2         0      262   0.48539 0.02097 0.72281
 ..................................................................
 0   127  312.00         1         3       16   0.11980 0.01846 2.12190
 0   128  326.00         1         1       14   0.11125 0.01902 2.19601
 0   129  332.00         1         2       11   0.10113 0.01980 2.29132
 0   130  350.00         1         1        9   0.08990 0.02054 2.40910
 0   131  428.00         0         8
Median Duration: 43.03
Duration times limited to: 428
Cases: 600  weighted: 600
```

columns show estimates of the survivor function, its standard errors, and the cumulated transition rate as defined above. If possible, an estimate of the median is calculated by linear interpolation of the survivor function. If the last episode is censored the survivor function does not go to zero; in this case the last observed time is printed at the end of the table.

Example 2 Command file `ple2.cf` (not shown) performs product-limit estimation with our main example data (`rrdat.1`). The command file is basically identical to `ltb1.cf` (see 6.5.1), we simply substitute the `ltb` command by

```
ple = ple.2;
```

Part of the output file, `ple.2`, is shown in Box 5.

Alternative Destination States. Product-limit estimation can easily be generalized for the case of single episodes with two or more different

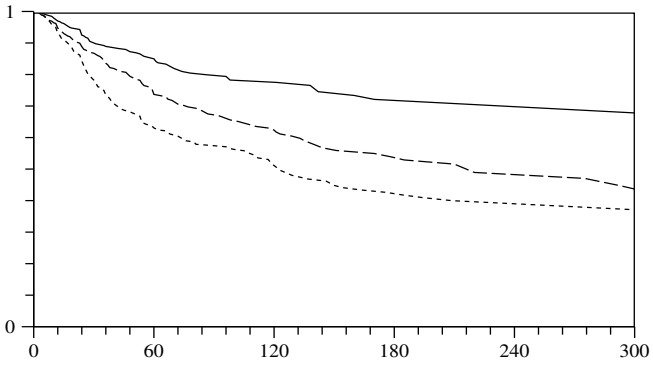


Figure 1 Plot of pseudo-survivor functions for three destination states, estimated with command file `ple3.cf`, based on our main example data. Plot created with command file `ple3p.cf`.

destination states. Again, one beings with N_j episodes having the same origin state j. Then, for each possible destination state $k \in \mathcal{D}_j$, one looks at the points in time, $\tau_{jk,l}$, where at least one transition to destination state k takes place. There are, say, $l = 1, \ldots, q_{jk}$ such points in time.

Let $E_{jk,l}$ denote the number of events at $\tau_{jk,l}$, and let $\mathcal{R}_{j,l}$ denote the risk set at the same point in time. Note that the risk set depends not on the destination state, it is defined as in the single transition case as the set of all episodes with origin state j, with a starting time less than $\tau_{jk,l}$, and with ending time equal to or greater than $\tau_{jk,l}$. The product-limit estimate of the pseudosurvivor functions may then be formally defined by

$$\tilde{G}_{jk}(t) = \prod_{l:\tau_{jk,l}<t} \left(1 - \frac{E_{jk,l}}{R_{j,l}}\right) \tag{5}$$

Obviously, a calculation of this estimate can use the same algorithm as in the single transition case. In the calculation for a specific destination state one only has to treat *all* episodes that do not end in this destination as if they are censored.

Example 3 Command file `ple3.cf` (also not shown) illustrates product-limit estimation with multiple destination states. This command file uses our main example data `rrdat.1` and specifies single episodes with three

alternative destination states. The command

```
ple = ple.3;
```

creates the output file `ple.3` containing three separate tables, one for each destination state. The resulting survivor functions are illustrated in Figure 1. (The command file for the plot is `ple3p.cf`, contained in the TDA example archive). When interpreting these curves, one should remember that they represent pseudo-survivor functions; the overall survivor function results from multiplication (not addition).

6.5.3 Survivor Function Quantiles

Output tables from the Kaplan-Meier procedure (see 6.5.2) are often very long. A shorter way to present most of the information is by tables that only show the estimated survivor function for some predefined quantiles. This is easily done with the `qo` option in the `ple` command. The syntax is

$$\texttt{qo = } o_1, o_2, \ldots, o_m\,\texttt{,}$$

where $o_1, o_2, \ldots, o_m$ are the orders of the quantiles. It is required that $1 > o_1 > o_2 > \ldots > o_m > 0$. Alternatively, one can use the syntax

$$\texttt{qo = } o_1(d)o_2\,\texttt{,}$$

to specify a sequence of orders with an increment d. Given this option in the `ple` command, for each transition found in the input data a table is printed summarizing the estimated survivor functions, or pseudo-survivor functions, by the requested quantiles. The quantiles are estimated by linear interpolation of the survivor function as given by the product-limit method.

Instead of `qo`, one can use the `qt` option with syntax

$$\texttt{qt = } t_1, t_2, \ldots, t_m\,\texttt{,}$$

The values t_i are then interpreted as time points and one gets tables with the corresponding orders. In this case, it is required that $0 \leq t_1 \leq t_2 \leq \ldots \leq t_m$.

When using the `qo` or `qt` option, the name of an output file in the `ple` command is optional. The tables containing the quantiles are always written into TDA's standard output.

Box 1 Table of quantiles, estimated with `ple3.cf`

SN	Org	Des	Survivor Function	Time Quantile
1	0	1	0.9000	30.0352
			0.8000	85.0602
			0.7000	233.8785
1	0	2	0.9000	16.8749
			0.8000	27.3090
			0.7000	41.6187
			0.6000	73.1997
			0.5000	122.2409
			0.4000	208.5655
1	0	3	0.9000	22.6799
			0.8000	47.1263
			0.7000	75.7539
			0.6000	131.5378
			0.5000	215.9553
			0.4000	321.9881

Example 1 To illustrate the `qo` option we use command file `ple3.cf` (see **6.5.2**) and add the command

```
ple (qo = 0.9 (0.1) 0.1) ;
```

The resulting table is shown in Box 1.

6.5.4 Comparing Survivor Functions

Basically two different methods are available to compare survivor functions. The first relies on the calculation of confidence intervals for each of the survivor functions and then to check if they are overlapping or not. This is possible both with the life table and the product-limit method. Both methods provide estimates of standard errors for the survivor function. Another possibility is to calculate specific test statistics to compare two or more survivor functions.

Defining Groups of Episodes. To make any comparisons there must be two or more groups of episodes. This can be done with the `grp` option in the `ple` command. The syntax is

```
grp = G1,G2,...,
```

with `G1,G2,...` names of variables contained in the data matrix. The current set of episodes is then split into m groups with m the number of indicator variables defined with the `grp` parameter. The first group, defined by `G1`, contains all episodes where the value of this variable is not zero, the second group is defined by `G2` in the same way, and so on.

Confidence Intervals. Using the `grp` option in the `ple` command, one gets a separate survivor function for each group. For each group the output table also contains the estimated standard errors, $\mathrm{SE}(\hat{G}(t))$, of the survivor function $\hat{G}(t)$. This allows the calculation of confidence intervals, based on the assumption that $(G(t) - \hat{G}(t))/\mathrm{SE}(\hat{G}(t))$ follows asymptotically a standard normal distribution. For standard 95 % confidence one would use

$$\hat{G}(t) \pm 1.96\,\mathrm{SE}\left(\hat{G}(t)\right)$$

Example 1 To illustrate the comparison of survivor functions by using confidence intervals, we continue with our main example data. Command file `ple5.cf` (contained in the TDA example archive) estimates survivor function separately for men and women. The resulting tables are written into an output file `ple.5`. Another command file, `ple5p.cf`, is then used to plot the two survivor functions and their confidence intervals.

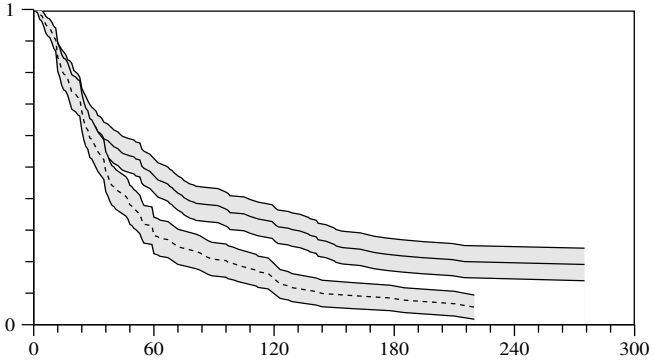


Figure 1 Survivor functions with confidence intervals for job durations of men and women, based on main example data `rrdat.1`. Plot created with command file `ple5p.cf`.

The resulting plot is shown in Figure 1. The survivor functions are very similar in the first two years (the time axis is given in months), but thereafter the job durations of women are significantly shorter. The median duration for men is 54.43 months, for women it is 35.61 months.

Construction of Test Statistics. Many different test statistics have been proposed to compare two or more survivor functions. We describes four which can be calculated with TDA; all are based on product-limit estimates of survivor functions.

It is assumed that m groups have been defined which are not intersecting. The whole sample is implicitly defined as the set of all episodes which are contained in one of these groups. Then, in exactly the same way as explained in connection with the product-limit method, all calculations are done for each transition in the whole sample separately. Therefore we only consider a sample of episodes which have the same origin state and are censored or have the same destination state. A sample, defined this way, consists of m groups and the following table can be calculated.

$$\begin{array}{llllllll} \tau_1 & R_{11} & E_{11} & R_{12} & E_{12} & \dots & R_{1m} & E_{1m} \\ \tau_2 & R_{21} & E_{21} & R_{22} & E_{22} & \dots & R_{2m} & E_{2m} \\ & & & & & \vdots & & \\ \tau_q & R_{n1} & E_{q1} & R_{q2} & E_{q2} & \dots & R_{qm} & E_{qm} \end{array} \tag{1}$$

These are the basic quantities for product-limit estimation, for the whole sample, and for each group separately. $\tau_1 < \tau_2 < \ldots < \tau_q$ are the points in time where at least one episode contained in the sample has an event. E_{lg} is the number of episodes contained in group g and having an event at τ_l; R_{lg} is accordingly defined as the number of elements in the risk set at τ_l for the episodes contained in group g, i.e., all episodes belonging to group g which have starting time less than τ_l and ending times equal to, or greater than, τ_l. Altogether, these quantities are sufficient for a product-limit estimation in each of the m groups.

Given this, the four test statistics can be defined, they are called S_ν ($\nu = 1, \ldots, 4$). Because the calculations only differ in different weights, we give their definition first. The weights are called $W_l^{(\nu)}$, and they are defined, for $l = 1, \ldots, q$, by

$$\begin{aligned} W_l^{(1)} &= 1 \\ W_l^{(2)} &= R_l \\ W_l^{(3)} &= \sqrt{R_l} \\ W_l^{(4)} &= \prod_{i=1}^{l} \frac{R_i - E_i + 1}{R_i + 1} \end{aligned} \tag{2}$$

The next step is to construct, for each of the four test statistics, one (m)-vector $U^{(\nu)}$ and one (m, m)-matrix $V^{(\nu)}$. The definitions are[1]

$$U_g^{(\nu)} = \sum_{l=1}^{q} W_l^{(\nu)} \left(E_{lg} - R_{lg} \frac{E_{l0}}{R_{l0}}\right) \tag{3}$$

$$V_{g_1 g_2}^{(\nu)} = \sum_{i=1}^{n} {W_l^{(\nu)}}^2 \frac{E_{l0}\left(R_{l0} - E_{l0}\right)}{R_{l0} - 1} \frac{R_{lg_1}}{R_{l0}} \left(\delta_{g_1 g_2} - \frac{R_{lg_2}}{R_{l0}}\right) \tag{4}$$

Finally, the test statistics are defined by

$$S_\nu = {U^{(\nu)}}' \, {V^{(\nu)}}^{-1} \, U^{(\nu)} \tag{5}$$

All of them follow, under the null hypothesis of no significant differences between the group-specific survivor functions, a χ^2-distribution with $m-1$ degrees of freedom. Note that, accordingly, the rank of $V^{(\nu)}$ is only

[1] δ_{ij} is the Kronecker symbol which is one, if $i = j$, and otherwise is zero.

Box 1 Comparing survivor functions with test statistics

```
Comparing survivor functions.

SN  Org Des   Test Statistic              T-Stat   DF   Signif
-----------------------------------------------------------------
 1    0   1    Log-Rank (Savage)          3.1227    1   0.9228
 1    0   1    Wilcoxon (Breslow)         2.6510    1   0.8965
 1    0   1    Wilcoxon (Tarone-Ware)     2.9767    1   0.9155
 1    0   1    Wilcoxon (Prentice)        3.0014    1   0.9168
```

$m - 1$. Therefore, one can use in the calculation of (5) a generalized inverse or, without loss of generality, one can omit the last dimension. TDA follows the latter of these two possibilities.[2]

Unfortunately, there is no uniform convention to name the different test statistics.[3] So we give the names used by TDA with some remarks about other naming conventions. In the order given by (2) we have:

1. Log-Rank (Savage). Other names are *Generalized Savage Test* (Andreß 1985, p. 158). The same name is used by BMDP, with *Mantel-Cox* added. SAS calculates this test statistics under the name *Logrank*.

2. Wilcoxon (Breslow). BMDP gives the name *Generalized Wilcoxon (Breslow)*, SAS uses only the label *Wilcoxon*.

3. Wilcoxon (Tarone-Ware). This test statistic was proposed by Tarone and Ware [1977] and is named accordingly. It is also calculated by BMDP, using the label *Tarone-Ware*.

4. Wilcoxon (Prentice). Finally, we have a test statistic explained by Lawless [1982, p. 423] with reference to Prentice [1978]. Since it is some type of a Wilcoxon test we use this name.

The `csf` option in the `ple` command (see **6.5.2**) can be used to request these test statistics. At least two groups must be specified with the `grp` parameter. An output file is optional.

Example 2 For an illustration of the `csf` option we use some biomet-

[2]If there are any more rank deficiencies in the V-matrices, TDA will report this in the standard output. If the `prot` option in the `ple` command is used to request a protocol file, the U-vectors and V-matrices are written into this protocol file.

[3]See Budde and Wargenau [1984] who give an overview with references to some software packages.

rical data published in Kalbfleisch and Prentice [1980, p. 2].[4] The data file is `rrdat.3`, the command file is `ple6.cf` (not shown). It uses the command

```
ple (csf,grp=Group1,Group2) ;
```

to request test statistics for a comparison of the survivor functions of the two groups. Part of the resulting output is shown in Box 1.

[4]These data are also used in some examples in the SAS User's Guide (Version 5, 1985, lifetest procedure).

6.5.5 State Distributions

A simple way of describing a set of multi-episode data is to calculate cross-sectional state distributions for a sequence of time points. This can be done with the `epsdat` command.

```
epsdat (
     t=...,          sequence of time points.
     ap=1,           append new data to end of output file.
) = fname;
```

The command requires a valid episode data structure and depends on whether single or multi-episode data are defined. The `t` parameter must be used to provide a sequence of nonnegative time points (for an explanation of the syntax see the `tp` parameter in 6.5.1.)

1. In the single episode case, the command calculates separately for each time point a frequency distribution of the origin states of those episodes which include the time point. (An episode includes a time point t if t is greater than, or equal to the starting time and less than the ending time.)

2. In the multi-episode case the command tries to find for each individual (defined by the `ID` parameter) and each time point an episode which includes the time point. Using then these episodes, the command calculates a frequency distribution of its origin states.

The resulting frequency distributions are written into the output file specified on the right-hand side of the command. If the `ap=1` parameter is used, the data are appended to the end of an existing file.

Box 1 Command file `ed5.cf`

```
nvar(

  dfile = ed1.dat,     # data file

  ID    [1.0] = c1,    # identification number
  SN    [1.0] = c2,    # spell number
  ORG   [1.0] = c3,    # origin state
  DES   [1.0] = c4,    # destination state
  TS    [2.0] = c5,    # starting time
  TF    [2.0] = c6,    # ending time
);
edef(
  ts = TS,    # starting time
  tf = TF,    # ending time
  org = ORG,  # origin state
  des = DES,  # destination state
# id  = ID,
# sn  = SN,
);
epsdat(         # write state distributions to d
  t = 0(5)30, # time points
) = d;
```

Example 1 To illustrate the `epsdat` command we use the data file `ed1.dat` (see **3.3.4**). The command file `ed5.cf` (Box 1) is set up for single episode data. By removing the comment characters in front of the `id` and `sn` parameter, it specifies multi-episode data. The `epsdat` command uses an abbreviation to define the sequence of time points

$$t = 0, 5, 10, 15, 20, 25, 30$$

The resulting output files, both for single and multi-episode data, are shown in Box 2. The first column shows the sequence of time points. Then follows, separately for each origin state in the input data, a column showing the number of episodes, or number of individuals, being in the corresponding state. The final column shows the number of episodes, or number of individual, without a valid state at the corresponding time point.

Box 2 Output from `epsdat` command

```
Singe episode data:

# State distributions.
# Time          1       2       3     Total  Missing
    0.0000      1       1       0         2        3
    5.0000      1       1       0         2        3
   10.0000      0       2       0         2        3
   15.0000      0       1       1         2        3
   20.0000      1       0       1         2        3
   25.0000      1       0       1         2        3
   30.0000      1       0       0         1        4

Multi-episode data:

# State distributions.
# Time          1       2       3     Total  Missing
    0.0000      1       1       0         2        0
    5.0000      1       1       0         2        0
   10.0000      0       2       0         2        0
   15.0000      0       1       1         2        0
   20.0000      1       0       1         2        0
   25.0000      1       0       1         2        0
   30.0000      1       0       0         1        1
```

6.6 Describing Sequence Data

This chapter describes some elementary commands that can be used to get information about a set of sequences. They require the specification of a sequence data structure as explained in **3.4.2**.

6.6.1 Sequence Length and Gaps

Given a sequence, let s_i denote the time point of the first valid state and t_i the time point of the last valid state. s_i will be called the (valid) starting time of the sequence, t_i its (valid) ending time, and $t_i - s_i + 1$ will be called the (valid) sequence length. Sequences may contain missing (negative) values. If they occur before the valid starting time, or after the valid ending time, they can normally be ignored. More problematic are missing values between the starting and ending time. They will be called *internal gaps*.

To provide basic information about starting and ending times, sequence length, and internal gaps, TDA provides the command `seqlg` with syntax shown in the following box.

```
seqlg (
    sn=...,      number of sequence data structure, def. 1
    sel=...,     expression for sequence selection
    v=...,       add variables ... to output file
    dtda=...,    request TDA description file
) = fname;
```

The right-hand side must provide the name of an output file, all other parameters are optional.

1. The `sn` parameter can be used to select a sequence data structure. As default, TDA uses the first of the currently defined sequence data structures.

2. The `sel` parameter can be used to select sequences. The syntax is

```
sel = expression,
```

TDA then only uses sequences (cases) where `expression` evaluates to a nonzero value.

3. The `v = varlist` parameter can be used to add the variables specified in `varlist` to the output file.

Box 1 Illustration of `seqlg` command with data file `seq.d1`

Case	starting time	ending time	sequence length	gap length	min gap length	max gap length	number of gaps
1	0	7	8	0	0	0	0
2	0	7	8	2	2	2	1
3	1	7	7	1	1	1	1
4	0	5	6	0	0	0	0
5	0	7	8	0	0	0	0

4. The `dtda` parameter can be used to request an additional output file containing an `nvar` command to read the data file created with the `seqlg` command.

Example 1 To illustrate the information written into the output file, we use the data file `seq.d1` (see **3.4.2**). Given that this data file has been used to define sequence data, the command `seqlg=d` creates the output file `d` shown in Box 1. The column "gap length" shows the total number of internal time points with a missing state. The command file in the example archive is `seq9.cf`.

6.6.2 General Characteristics

To get some general information about a sample of sequences one can use the `seqgc` command with syntax shown in the following box.

```
seqgc (
    sn=...,      number of sequence data structure, def. 1
    sel=...,     expression for sequence selection
    v=...,       add variables ... to output file
    dtda=...,    request TDA description file
) = fname;
```

The right-hand side must provide the name of an output file. All other parameters are optional and have the same meaning as explained in **6.6.1** for the `seqgl` command.

To illustrate the information written into the output file, we use the data file `seq.d1` (see **3.4.2**). Given that this data file has been used to define sequence data, the command

```
seqgc (dtda = t) = d;
```

creates an output file, `d`, and an additional description file, `t`, both shown in Box 1. (The command file in the example archive is `seq9.cf`.)

1. The first column contains the case number, written with a 6.0 print format. (All other entries use a 4.0 print format.)
2. The second column records the individual sequence length, defined as the number of time points from the first valid state until the last valid state, that is, the *valid sequence length*. Of course, there might be missing states in between.
3. It follows an information about the number of different states in the individual sequences, not counting missing states.
4. Then comes the number of state changes (events), calculated for the valid sequence length and including changes from and into a missing state.

Box 1 Illustration of `seqgc` command with data file `seq.d1`

```
 1    8    3    4    1    4    3    0    1    2    2    0
 2    8    3    4    2    3    1    2    1    2    1    1
 3    7    2    2    4    2    0    1    1    1    0    1
 4    6    2    1    2    4    0    0    1    1    0    0
 5    8    3    4    1    4    3    0    1    2    2    0

nvar(
  dfile = d,
  noc = 5,
  CASE <5>[8.0] = c1 , # case number
  SLEN <2>[4.0] = c2 , # sequence length
  NDS  <2>[4.0] = c3 , # number of different states
  NEV  <2>[4.0] = c4 , # number state changes
  DUR1 <2>[4.0] = c5 , # duration in state 1
  DUR3 <2>[4.0] = c6 , # duration in state 3
  DUR7 <2>[4.0] = c7 , # duration in state 7
  DURM <2>[4.0] = c8 , # duration in missing state
  NEP1 <2>[4.0] = c9 , # number of episodes in state 1
  NEP3 <2>[4.0] = c10, # number of episodes in state 3
  NEP7 <2>[4.0] = c11, # number of episodes in state 7
  NEPM <2>[4.0] = c12, # number of episodes in missing state
);
```

5. The next entries show the number of time points, calculated for the valid sequence length, that the individual sequence stays in the different states.

6. Then follows the total duration of internal gaps.

7. The next entries show the number of episodes, differentiated according to their origin state.

8. Then follows the number of internal gaps.

9. And finally, any variables specified with the `v` parameter are added to the individual records.

6.6.3 State Distributions

To get information about the cross-sectional state distributions in a sample of sequences, one can use the `seqsd` command with syntax shown in the following box.

```
seqsd (
    sn=...,      number of sequence data structure, def. 1
    sel=...,     expression for sequence selection
    dtda=...,    request TDA description file
) = fname;
```

The right-hand side must provide the name of an output file. All other parameters are optional and have the same meaning as explained in 6.6.1 for the `seqgl` command.

Example 1 To illustrate the information written into the output file, we use the data file `seq.d1` (see 3.4.2). Given that this data file has been used to define sequence data, the command

```
seqsd (dtda = t) = d;
```

creates an output file, `d`, and an additional description file, `t`, both shown in Box 1. The command file in the example archive is `seq9.cf`.

Homogeneity of State Distributions. The homogeneity of state distributions can be assessed by a simple entropy measure (see, e.g., Theil [1972]). Let N_t denote the number of individuals with a valid state at time t, and p_{tj} the proportion of N_t being in state j. Then, assuming states $j = 1, \ldots, q$, the entropy is defined by

$$E_t = -\sum_{j=1}^{q} p_{tj} \log(p_{tj})$$

with the convention that $0 \log(0) = 0$. It follows that $0 \leq E_t \leq \log(q)$. The entropy becomes zero if all individuals are in the same state, it takes

Box 1 Illustration of `seqsd` command with data file `seq.d1`

```
 0      1      1      2      4      1      5
 1      1      4      0      5      0      5
 2      0      4      1      5      0      5
 3      3      1      0      4      1      5
 4      2      3      0      5      0      5
 5      1      3      0      4      1      5
 6      1      0      2      3      2      5
 7      1      1      2      4      1      5

nvar(
  dfile = d,
  noc = 8,
  TIME  <2>[6.0] = c1, # time
  NST1  <2>[6.0] = c2, # cases in state 1
  NST3  <2>[6.0] = c3, # cases in state 3
  NST7  <2>[6.0] = c4, # cases in state 7
  VALID <2>[6.0] = c5, # cases in valid states
  NMISS <2>[6.0] = c6, # cases in missing state
  TOTAL <2>[6.0] = c7, # total number of cases
);
```

Box 2 Syntax for `seqen` command

```
seqen (
    sn=...,      number of sequence data structure, def. 1
    sel=...,     expression for sequence selection
    fmt=...,     print format, def. 10.4
    dtda=...,    request TDA description file
) = fname;
```

its maximum value if the individuals are equally spread over the different states.

To calculate these entropy measures, TDA provides the command `seqen` with syntax shown in Box 2. The right-hand side must provide the name of an output file. All other parameters are optional and have the same meaning as explained in 6.6.1 for the `seqgl` command.

Box 3 Illustration of `seqen` command, based on `seq.d1`

```
# Time       N     Entropy
     0       4      1.0397
     1       5      0.5004
     2       5      0.5004
     3       4      0.5623
     4       5      0.6730
     5       4      0.5623
     6       3      0.6365
     7       4      1.0397
```

Example 2 To illustrate the `seqen` command, we use again the data file `seq.d1` (see **3.4.2**). Given that this data file has been used to define sequence data, output of the `seqen` command is shown in Box 3. The command file in the example archive is `seq9.cf`.

6.6.4 Pattern Matching

Given a definition of sequence data, the `seqpm` command can be used to find patterns in the sequences. The syntax is shown in the following box.

```
seqpm (
    sn=...,          number of sequence data structure, def. 1
    ps=...,          definition of patterns
    df=...,          test output file
    nfmt=...,        integer print format, def. 4
    v=...,           additional variables for output file
    dtda=...,        TDA description for output file
) = output_file_name;
```

The right-hand side must provide the name of an output file and the `ps` parameter must be used for a definition of patterns. All other parameters are optional. Patterns must be defined in the following way:

```
ps = [a1,a2,...],[b1,b2,...],...,
```

where each pair of square brackets specifies one pattern. The entries, `a1,a2,...` and so on, can be one of the following:

1. Non-negative integers for valid states.
2. The character '?' matches any single state.
3. The character '*' matches any sequence of states.
4. The character '+' matches any repeat of the previous state.
5. The character '-' matches any sequence of identical states.

Note that the command only recognizes sequences that do not contain missing values (negative state numbers).

The `seqpm` command creates an output file that will contain one record for each sequence containing the following information:

Box 1 Command file `seqpm1.cf`

```
nvar(
    dfile = seq.d1,
    ID = c1,
    Y0 = c2,
    Y1 = c3,
    Y2 = c4,
    Y3 = c5,
    Y4 = c6,
    Y5 = c7,
    Y6 = c8,
    Y7 = c9,
);
seqdef = Y0,,Y7;
seqpm(
    ps = [-],[3,3],[3,*,3],
    df = df,
    dtda = t,
) = d;
```

Box 2 Output file from command file `seqpm1.cf`

```
ID   LEN  P1   P2   P3
----------------------
 1    8    5    2    1
 4    6    2    2    1
 5    8    5    2    1
```

1. The sequence case id number.
2. The length of the sequence.
3. Finally, for each pattern, the number of matches.

In addition, one can add any variables to the output file with the

```
v = varlist,
```

parameter. Also, one can request a TDA description of the output file with the `dtda` parameter.

Example 1 To illustrate the `seqpm` command we use data file `seq.d1` as described in **3.4.2**. The command file is shown in Box 1. The resulting output file is shown in in Box 2.

6.7 Investigating Proximities

This chapter is intended to deal with proximity data. There is currently only a single section discussing the calculation of proximity measures for sequence data.

6.7.2 Sequence Proximity Measures

6.7.2 Sequence Proximity Measures

Regression models for events aim to describe the evolution of sequences by focusing on transitions. As a complementary strategy, one can try to describe whole sequences. This can be done in two different ways. One can search for some interesting patterns in each sequence separately; or one can compare all sequences in a given sample. In any case, one needs a proximity measure to assess the similarity of sequences, or of sequences with some pattern.

This chapter discusses a family of proximity measures for sequences based on "optimal matching". Basic information about this approach can be found in Sankoff and Kruskal [1983]. Bock [1984] provides a short introduction. A very useful source of mathematical insights and algorithms is Waterman [1995]. For a discussion of sociological applications, see Abbott [1983], [1995], Abbott and Hrycak [1990].

6.7.2.1 Optimal Matching

We consider sequences of states which are elements of a finite state space, say $\mathcal{Y}$. $\mathcal{S}$ denotes the set of all finite sequences over $\mathcal{Y}$, meaning that

$$\text{if } a \in \mathcal{S} \text{ then } a = (a_1, \ldots, a_n) \text{ with } a_1, \ldots, a_n \in \mathcal{Y}$$

$n = |a|$ is the length of the sequence. We want to compare two sequences $a, b \in \mathcal{S}$, possibly with different lengths. The basic idea is to define some elementary operations which can be used to sequentially transform one sequence until it becomes equal to the other sequence. Let Ω denote the set of basic operations and $a[\omega]$ the sequence resulting from a by applying the operation $\omega \in \Omega$. Most applications consider just three elementary operations:

- Insertion: $a[\iota]$ denotes the sequence resulting from $a \in \mathcal{S}$ by inserting one new element (a state from $\mathcal{Y}$) into the sequence a.
- Deletion: $a[\delta]$ denotes the sequence resulting from a by deleting one element from this sequence.
- Substitution: $a[\sigma]$ denotes the sequence resulting from a by changing one of its elements into another state.

Of course, we can think of sequentially applying elementary operations to a given sequence. Let $a[\omega_1, \omega_2, \ldots, \omega_k]$ denote the new sequence resulting from a by applying first the elementary operation ω_1, then ω_2, and so on until finally ω_k. Then, given two sequences $a, b \in \mathcal{S}$, we can ask for a sequence of elementary operations which transforms a into b.

In general, there will be many such sequences of elementary operations which transform a into b. Now, the intuitive idea for developing a distance measure for sequences is to look for the shortest sequence of elementary operations which transform a into b. A slightly more general approach is to evaluate the elementary operations by introducing $c(\omega)$ = cost of applying the elementary operation $\omega \in \Omega$. We will assume that $0 < c(\omega) < \infty$. The cost of applying a sequence of elementary operations will be denoted by

$$c[\omega_1, \omega_2, \ldots, \omega_k] = \sum_{i=1}^{k} c[\omega_i]$$

Setting $c[\,] = 0$ for no operation, we can formally define

$$d_\Omega(a,b) = \min\left\{\, c[\omega_1,\ldots,\omega_k] \,|\, b = a[\omega_1,\ldots,\omega_k], \omega_i \in \Omega, k \geq 0 \,\right\}$$

to measure the distance between the sequences a and b. This measure is by definition nonnegative, and $d_\Omega(a,b) = 0$ only if $a = b$. Symmetry does not automatically hold, but can be forced by equating insertion and deletion costs, or by a slightly different definition: minimum cost of transforming a into b *or* b into a. Whether the distance measure will also be transitive, and thus constitutes a metric distance, depends on the definition of the set of elementary operations, Ω. Transitivity is automatically guaranteed for the simple case when there are only insertions, deletions, and substitutions.

"Optimal matching" without further qualification normally means referring to this simple distance measure based on $\Omega = \{\iota, \delta, \sigma\}$ resulting in a metric distance. Of course, the distance measure also depends on the definition of the cost functions $c[\omega]$. These cost functions can be arbitrarily defined with respect to the intended application. As a special case, one can set

$$c[\iota] = c[\delta] = 1 \quad \text{and} \quad c[\sigma] = 2$$

The distance between two sequences a and b is then simply the number of indel operations (insertions and deletions) which are necessary to transform one sequence into the other. For the procedure described in **6.7.2.2**, this will be the default cost functions if not otherwise specified by the user.

The set of elementary operations, Ω, can be extended in many different ways. However, the current implementation in TDA offers only one, but possibly important, extension: so-called indel functions that allow deleting or inserting a series of consecutive states, see **6.7.2.4**.

Basic Algorithm. If the elementary operations consist only of insertions, deletions and substitutions, the distance measure can be calculated with a simple dynamic programming method. To explain this method, we follow Kruskal and Sankoff [1983], p. 266).[1]

Let $\mathcal{Y}$ denote the finite state space and ϕ an "empty state" which is

[1] See also Waterman [1995], pp. 192–194, and for an illustration of the algorithm, Abbott and Hrycak [1990].

not contained in $\mathcal{Y}$. The cost functions are

$$w(x,y) = \text{substitution cost},\ x,y \in \mathcal{Y}$$
$$w(x,\phi) = \text{deletion cost},\ x \in \mathcal{Y}$$
$$w(\phi,y) = \text{insertion cost},\ y \in \mathcal{Y}$$

The expression

$$\begin{bmatrix} x_1 & \cdots & x_p \\ y_1 & \cdots & y_p \end{bmatrix} \tag{1}$$

is called an *alignment* if $x_i, y_i \in \mathcal{Y} \cup \{\phi\}$ and there is no column with $x_i = y_i = \phi$. The *length* of an alignment (i.e., the distance between the two sequences) is defined by $\Sigma_i\, w(x_i, y_i)$.

Now, let $a = (a_1, \ldots, a_m)$ and $b = (b_1, \ldots, b_n)$ be two sequences with states in $\mathcal{Y}$. We say that (1) is an alignment between the sequences a and b if by inserting empty states into a it can be made equal to $x = (x_1, \ldots, x_p)$ and in the same way b can be made equal to $y = (y_1, \ldots, y_p)$. Given these definition, we can finally define the (standard) distance between a and b, denoted $d(a,b)$, as the minimum possible length of any alignment of these two sequences. Additional notation to describe the algorithm is

$$a^i = (a_1, \ldots, a_i)\,,\ b^j = (b_1, \ldots, b_j)\,,\ \text{and}\ d_{ij} = d(a^i, b^j)$$

Using this notation, we need to calculate $d(a,b) = d_{mn}$. This can be done by calculating the elements of the $(m+1, n+1)$ matrix

$$D = \left(d_{ij}\right) \quad i = 0, \ldots, m;\ j = 0, \ldots, n \tag{2}$$

recursively in the following way. The first step is initializing the first row and first column of this matrix:

$$\begin{aligned} d_{0,0} &= 0 \\ d_{0,j} &= d_{0,j-1} + w(\phi, b_j) \quad j = 1, \ldots, n \\ d_{i,0} &= d_{i-1,0} + w(a_i, \phi) \quad i = 1, \ldots, m \end{aligned}$$

All other elements of D can then be calculated by using three predecessors. The recurrence relation is:

$$d_{ij} = \min \begin{cases} d_{i-1,j} + w(a_i, \phi) \\ d_{i-1,j-1} + w(a_i, b_j) \\ d_{i,j-1} + w(\phi, b_j) \end{cases}$$

Having calculated all elements of D, one finally finds the required distance in the element d_{mn}.

Computation time for this algorithm, and storage requirements, are proportional to mn. In the TDA implementation, the matrix D is defined for single precision floating point numbers, and the required memory is $4L^2$ bytes, L being 1 plus the maximum sequence length in the input data.

6.7.2.2 The `seqm` Command

The TDA command to request optimal matching of sequences is `seqm`, with syntax shown in Box 1. The right-hand side must provide the name of an output file. All other parameters are optional.

Selecting alignments. There are two basic options which can be selected with the `m` parameter.

1. If `m=1` (default), the `seqm` command uses a single sequence data structure. If not otherwise specified with the `sn` parameter, the first of the currently defined sequence data structures will be used. As default, an alignment for each pair of sequences in the selected sequence data structure is calculated. If there are n sequences, there will be $n(n-1)/2$ alignments, and the output file will contain one record for each alignment.

There are two options to restrict the number of alignments. The `r` parameter can be used to request a random selection of alignments (see 6.7.2.6). Another possibility is to use only some specified sequences for comparison. This can be requested with the parameter

`cn =` $i_1, i_2, \ldots, i_m$`,`

where $i_1, i_2, \ldots, i_m$ are the case numbers of some sequences in the data matrix. Then, all sequences of the selected sequence data structure are only compared with these selected sequences resulting in a total number of nm alignments. In this case, the output file will contain n records, each containing m distances.

2. Alternatively, one can use the `m=2` option for pairwise optimal matching of sequences taken from two sequence data structures. In this case, the parameter

`sn =` k_1, k_2`,`

must be used to specify the two sequence data structures, and it is required that the two sequence data structures have the same state space (see 6.7.2.7).

Preprocessing sequences. Before alignment of any two sequences, they are shifted to a common process time axis: $t = 1, 2, 3, \ldots, L$, where

Box 1 Syntax for `seqm` command

```
seqm (
    m=...,        selection of method, def. 1
    sn=...,       selection of sequence data structure(s), def. 1
    icost=...,    indel cost specification
    scost=...,    substitution cost specification
    rr=1,         use common sequence length
    sm=...,       option for preprocessing sequences
    r=...,        random selection of sequences
    s=...,        print sequence of distances, or LCS
    cn=...,       compare with specified sequences
    max=...,      alignment restriction
    tfmt=...,     print format for distances, def. 5.2
    v=...,        add variables ... to output file
    dtda=...,     create TDA description file
    df=...,       create test output file
    tst=...,      additional test options
    fmt=...,      print format for test output file
) = fname;
```

L is the maximum sequence length. As default, only sequences with a positive length and without internal gaps will be compared. This can be changed with the `sm` parameter. There are three options:

- `sm=1` Internal gaps will be disregarded, that is, they will simply be skipped in preparing a sequence for comparison with another one.
- `sm=2` Each contiguous subsequence of identical states is substituted by a single occurrence of that state.
- `sm=1,2` This selects `sm=1` and `sm=2`.

The `tst` parameter (explained below) can be used to check whether the selected option has the desired effect.[1]

As mentioned in **6.7.2.1**, optimal matching does not require that sequences have equal length. As an option (`rr=1`), one can restrict the

[1]Of course, any other preprocessing of sequences can be done before inputting the data to the program, or using TDA's operators to change variables. For example, Dijkstra and Taris [1995] have proposed a method which only takes into account states that appear in both sequences to be compared. Whether this makes sense is questionable, see the comments by Abbott [1995a].

alignment to the common sequence length. Then, if l_i is the length of sequence i and l_j is the length of sequence j, the alignment procedure uses only the common sequence length $\min\{l_i, l_j\}$.

Specifying indel cost. The default distance measure is $d(a, b)$, based on a set of elementary functions, Ω, only consisting of insertions, deletions, and substitutions. Insertion and deletion costs are always identical and called *indel cost*. The default indel cost is 1, and the default substitution cost is 2. The alignment, then, amounts to calculating the longest common subsequence (LCS).

Indel cost can be changed with the `icost` parameter. There are three possible ways to specify alternative definitions. Using

`icost =` α`,`

with some nonnegative real value α, indel cost will be equal to α. Alternatively, indel costs can be specified as a sequence of values for the common time axis. This option requires that the user first defines a $(1, L)$ or $(L, 1)$ matrix containing the indel cost values; L being the maximal sequence length in the selected sequence data structure.[2] Assuming that such a matrix has been defined and named `M`, one can use the parameter

`icost = M,`

to request using the values from this matrix.

As a third possibility, one can extend the set of operations, Ω, to allow for simultaneously inserting or deleting contiguous subsequences consisting of one or more states, based on a linear indel cost function

$$g(l) = \alpha + \beta\,(l - 1) \quad \text{with} \quad \alpha, \beta \geq 0$$

l being the length of the subsequence. The extension will be effective if the user specifies this cost function with the parameter

`icost =` α, β`,`

This extension will be explained and illustrated in **6.7.2.4**. Note that the `max` parameter to restrict the possible alignments is not effective when using an extended distance measure.

Specifying substitution cost. Given two sequences $a = (a_1, \ldots, a_m)$ and $b = (b_1, \ldots, b_n)$, default substitution cost is $w(a_i, b_j) = 2$, if $a_i \neq b_j$;

[2] See **5.1.2** for commands to define matrices.

substitution cost for identical states is always zero. Some alternative definitions can be selected with the parameter

```
scost = n,
```

If $n = 1$, substitution cost is calculated as the absolute difference $w(a_i, b_j) = |a_i - b_j|$. If $n = 2$, substitution cost will be derived from the frequency of transitions in the sequences, see **6.7.2.5**.

Alternatively, one can define a complete substitution cost matrix with the `mdef` command for defining matrices, see **5.1.2**. This must be a (n_s, n_s) matrix, n_s being the number of states in the selected sequence data structure. Given that such a matrix $M = (m_{ij})$ has been defined, one can use the parameter

```
scost = M,
```

to request that this matrix shall be used, meaning that $w(a_i, b_j) = m_{ij}$. Remember that the states occurring in a sequence data structure are always sorted in ascending order and then mapped to internal state numbers $0, 1, \ldots, n_s - 1$. The ordering of substitution cost values in the matrix `M` must correspond to these internal state numbers. In any case, one should check the specification of indel and substitution cost with the test output option explained below.

Restricted alignment. The `max` parameter can be used to restrict the alignment procedure. This option will be explained in **6.7.2.6**.

Output options. By default, the output file will contain a separate record for each alignment. If n is the number of sequences, the number of records will be $n(n-1)/2$, if `m=1`, and n, if `m=2`. This might be different if the `r` parameter is used to request a random selection of alignments, or if the `cn` parameter is used to request alignments with some fixed reference sequences.

By default, each record of the output file will contain the case numbers of the sequences, their length and the value of the distance measure. The print format for the distance measure can be controlled with the `tfmt` parameter, default is `tfmt=5.2`.

The `s` parameter can be used to modify the printing of distances. If `s=0` (default), only the final distance, $d(a, b)$, between any two sequences $a = (a_1, \ldots, a_m)$ and $b = (b_1, \ldots, b_n)$ is written into the output file. Alternatively, with `s=1`, one gets all sequentially calculated distances

$d(a^i, b^j)$ (see the definition in **6.7.2.1**), meaning that each record of the output file will contain the sequential distances

$$\begin{aligned} d(a^i, b^i) &\quad \text{for} \quad i = 1, \ldots, \min\{m, n\} \\ d(a^m, b^j) &\quad \text{for} \quad j = m+1, \ldots, n \\ d(a^i, b^n) &\quad \text{for} \quad i = n+1, \ldots, m \end{aligned}$$

This option is useful for sequentially comparing sequences. The sequence of distances, $d(a^i, b^i)$, can be interpreted as showing how the distance between a and b evolves over time.

When using default cost functions, one can alternatively request a printing of the longest common subsequences with the `s=2` parameter.

In addition, one can use the `v=varlist` parameter to add variables to the output data file. Values of the variables specified in `varlist` will be written into the output file for both sequences which take part in the alignment. To see which variables have been written into the output file, one can request an additional data description file with the `dtda` parameter.

Test output. The `df` parameter can be used to request an additional test output file. By default, this file will provide information about the set up of indel and substitution cost. Additional information can be requested with the parameter

`tst =` $i, j, \ldots,$

where $i, j, \ldots$ are integers in the range $1, \ldots, 3$. The meaning is as follows.

1 This option prints the initial D matrix into the test output file.

2 This option prints the sequences into the test output file. This should be helpful in checking whether the `sm` parameter has the desired effect.

3 Using this option, for each alignment of two sequences, the matrix D, defined in (2) above, is written into the test output file. (Additional matrices will be written into the test output file when working with an extended distance measure.) Illustration will be given in **6.7.2.3**.

6.7.2.3 Examples

This section illustrates the `seqm` command with some examples. We use very small example data files in order to easily check what the command is doing.

Example 1 We begin with the four sequences shown in Box 1. A command file, `seqm1.cf`, is shown in Box 2. This command file first reads the data file and defines a sequence data structure and then requests optimal matching with the `seqm` command. Only the `dtda` parameter is used to request a description of the output file. All other parameters have default values.

Standard output from the command file is shown in Box 3. The command has performed pairwise alignment of the four sequences using default indel and substitution cost. The command has created two files, a data file (`seqm1.d`) and a description file (`seqm1.tda`) that describes the variables in the data file. Both files are shown in Box 4.

To add variables to the output file, one can use the `v` parameter. For example, if adding the parameter `v=Y1,,Y5` to the `seqm` command in `seqm1.cf`, the output file would contain 10 additional variables (the variables `Y1,...,Y5` for both sequences).

Box 1 shows a second data file, `seqm.d1a`, which is basically identical with `seqm.d1` but where the sequences are differently placed on the time axis. Since, as default, TDA uses only sequences without internal gaps and skips leading missing values, using this second data file will give identical results when used for optimal matching. (See the command file `seqm1a.cf` in the example archive.)

The data file `seqm1.d` in Box 4 only shows the final distance for each alignment. The `s=1` option can be used to get information about how the distances evolve during process time. Adding this parameter to the `seqm` command would result in the output file shown in Box 5.

Box 1 Example sequence data files

```
Data file: seqm.d1      Data file: seqm.d1a
0 0 0 0 0               -1  0  0  0  0  0 -1 -1 -1
0 0 1 0 0               -1 -1  0  0  1  0  0 -1 -1
0 0 0 1 0               -1 -1 -1  0  0  0  1  0 -1
0 0 1 1 0               -1 -1 -1 -1  0  0  1  1  0
```

Box 2 Command file `seqm1.cf`

```
nvar(
    dfile = seqm.d1,
    Y{1,5} = c1,
);
seqdef = Y1,,Y5;
seqm(
    dtda = seqm1.tda,
) = seqm1.d;
```

Box 3 Standard output from `seqm1.cf`

```
Sequence proximity measures. Current memory: 260732 bytes.
Optimal matching.
Using sequence data structure 1.
Number of states: 2. Max sequence length: 5

Default indel cost: 1.
Default substitution cost: 2.

Starting alignment procedure.
Number of sequences (cases): 4
Sequences with zero length or internal gaps: 0
Sequences used for alignment: 4

Number of alignments: 6
6 record(s) written to output file: seqm1.d
Maximum distance between sequences 4 and 1: 4
TDA description written to: seqm1.tda
```

Box 4 Output files created by `seqm1.cf`

```
Output file: seqm1.d
----------------------------
 2      1      5      5  2.00
 3      1      5      5  2.00
 3      2      5      5  2.00
 4      1      5      5  4.00
 4      2      5      5  2.00
 4      3      5      5  2.00

TDA description file: seqm1.tda
-------------------------------
nvar(
  dfile = seqm1.d,
  noc = 6,
  SEQ1N <5>[6.0] = c1  , # sequence A case number
  SEQ2N <5>[6.0] = c2  , # sequence B case number
  SEQ1L <2>[6.0] = c3  , # sequence A length
  SEQ2L <2>[6.0] = c4  , # sequence B length
  DIST  <4>[5.2] = c5  , # distance
);
```

In the first example we have used default indel and substitution cost and optimal matching is then basically identical to calculating the longest common subsequences (LCS). This allows to use the `s=2` parameter to request a printing of the LCS. To illustrate, adding the parameter `s=2` to command file `seqm1.cf` will create the output files shown in Box 6.

The additional variables in the output file show the length of the LCS and its values. If the LCS is shorter than the maximal sequence length, the remaining columns are filled with missing values (-1). Note the simple relationship

$$\text{length of longest common subsequence } = \frac{1}{2}\left(m + n - d(a,b)\right)$$

where $d(a,b)$ the the optimal matching distance when using default cost functions; m and n denote the length of the first and second sequence, respectively.

Box 5 Illustration of `s=1` option (`seqm1b.cf`)

```
Data file
-------------------------------------------------------
 2      1      5      5  0.00  0.00  2.00  2.00  2.00
 3      1      5      5  0.00  0.00  0.00  2.00  2.00
 3      2      5      5  0.00  0.00  2.00  2.00  2.00
 4      1      5      5  0.00  0.00  2.00  4.00  4.00
 4      2      5      5  0.00  0.00  0.00  2.00  2.00
 4      3      5      5  0.00  0.00  2.00  2.00  2.00

Description file
-------------------------------------------------------
nvar(
  dfile = ...,
  noc = 6,
  SEQ1N <5>[6.0] = c1   , # sequence A case number
  SEQ2N <5>[6.0] = c2   , # sequence B case number
  SEQ1L <2>[6.0] = c3   , # sequence A length
  SEQ2L <2>[6.0] = c4   , # sequence B length
  DIST1 <4>[5.2] = c5   , # distance, time 1
  DIST2 <4>[5.2] = c6   , # distance, time 2
  DIST3 <4>[5.2] = c7   , # distance, time 3
  DIST4 <4>[5.2] = c8   , # distance, time 4
  DIST5 <4>[5.2] = c9   , # distance, time 5
);
```

Box 6 Illustration of `s=2` option (`seqm1c.cf`)

```
Data file
-------------------------------------------------
 2      1      5      5  2.00    4  0  0  0  0 -1
 3      1      5      5  2.00    4  0  0  0  0 -1
 3      2      5      5  2.00    4  0  0  1  0 -1
 4      1      5      5  4.00    3  0  0  0 -1 -1
 4      2      5      5  2.00    4  0  0  1  0 -1
 4      3      5      5  2.00    4  0  0  1  0 -1

Description file
-------------------------------------------------
nvar(
  dfile = ...,
  noc = 6,
  SEQ1N <5>[6.0] = c1  , # sequence A case number
  SEQ2N <5>[6.0] = c2  , # sequence B case number
  SEQ1L <2>[6.0] = c3  , # sequence A length
  SEQ2L <2>[6.0] = c4  , # sequence B length
  DIST  <4>[5.2] = c5  , # distance
  LCSL  <2>[4.0] = c6  , # length of LCS
  LCS1 <2>[2.0] = c7  , # LCS t=1
  LCS2 <2>[2.0] = c8  , # LCS t=2
  LCS3 <2>[2.0] = c9  , # LCS t=3
  LCS4 <2>[2.0] = c10 , # LCS t=4
  LCS5 <2>[2.0] = c11 , # LCS t=5
);
```

Box 7 Sequence data file `seqm.d2`

```
-1  1  1 -1  2
 2  2 -1 -1  2
 1  1  2  2 -1
```

Box 8 Command file `seqm2.cf`

```
nvar(
    dfile = seqm.d2,
    Y{1,5} = c1,
);
seqdef = Y1,,Y5;
seqm(
    sm = 1,
    df = seqm2.tst,
    tst = 2,
    dtda = seqm2.tda,
) = seqm2.d;
```

Example 2 To illustrate the preprocessing of sequences, we use the data file `seqm.d2` in Box 7. The first two sequences contain internal gaps. Without preprocessing these sequences, there would be only one valid sequence and the `seqm` command would perform no alignments.

As explained in **6.7.2.2**, there are two preprocessing options. The first one, `sm=1`, simply skips internal gaps. This has been done with command file `seqm2.cf` in Box 8. This command file also requests a test output file, `seqm2.tst`, shown in Box 9. This file shows the set up of cost functions (in this example we have again default indel and substitution cost), and since we have used the `tst=2` option, it also shows the sequences actually used for alignment. In this example, there are three pairs of sequences for three alignments.

The lower part of Box 9 illustrates how the sequences would look like if we had used both preprocessing options, `sm=1,2`. Each consecutive subsequence of identical states is then substituted by a single occurrence of that state.

Box 9 Test output file created by `seqm2.cf`

```
Optimal matching test output file.
Number of states: 2
Max sequence length: 5

Indel cost
1 1 1 1 1

Substitution cost
0 2
2 0

Sequence A (case number 2)
  2  2  2
Sequence B (case number 1)
  1  1  2

Sequence A (case number 3)
  1  1  2  2
Sequence B (case number 1)
  1  1  2

Sequence A (case number 3)
  1  1  2  2
Sequence B (case number 2)
  2  2  2

Sequences used for alignment if sm=1,2
--------------------------------------
Sequence A (case number 2)
  2
Sequence B (case number 1)
  1  2

Sequence A (case number 3)
  1  2
Sequence B (case number 1)
  1  2

Sequence A (case number 3)
  1  2
Sequence B (case number 2)
  2
```

Box 10 Sequence data file `seqm.d3`

```
4 5 6 7 7 7 7  7
1 1 2 3 4 5 6 -1
```

Box 11 Command file `seqm3.cf`

```
nvar(
    dfile = seqm.d3,
    Y{1,8} = c1,
);
seqdef = Y1,,Y8;

mdef(SCOST,10,10) =
  0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
  0.1, 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
  0.2, 0.1, 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
  0.3, 0.2, 0.1, 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
  0.4, 0.3, 0.2, 0.1, 0.0, 0.1, 0.2, 0.3, 0.4, 0.5,
  0.5, 0.4, 0.3, 0.2, 0.1, 0.0, 0.1, 0.2, 0.3, 0.4,
  0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0, 0.1, 0.2, 0.3,
  0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0, 0.1, 0.2,
  0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0, 0.1,
  0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0;

seqm(
    icost = 1,
    scost = SCOST,
    df = seqm3.tst,
    tst = 2,3,
    dtda = seqm3.tda,
) = seqm3.d;
```

Example 3 To illustrate user-defined cost functions we use an example given by Abbott and Hrycak ([1990], p. 179). The data file, `seqm.d3` (Box 10), consists of two sequences.

Box 11 shows the command file, `seqm3.cf`. Indel cost is simply 1 (actually the default); substitution cost is defined by a matrix, `SCOST`. One should note how this matrix is used. The state space for the two sequences is $\{1, 2, 3, 4, 5, 6, 7\}$. These states are mapped to internal state number $\{0, 1, 2, 3, 4, 5, 6\}$. The matrix element $\mathrm{SCOST}(i, j)$ corresponds to the internal state numbers $i - 1$ and $j - 1$.

Command file `seqm3.cf` also requests a test output file with options `tst=2,3`. Part of this test output file is shown in Box 12. The files shows

Box 12 Part of test output file from `seqm3.cf`

```
Sequence A (case number 2)
1  1  2  3  4  5  6
Sequence B (case number 1)
4  5  6  7  7  7  7  7

D Matrix
           0     1     2     3     4     5     6     7     8
      ------------------------------------------------------
           B     4     5     6     7     7     7     7     7
      ------------------------------------------------------
0   A | 0.00  1.00  2.00  3.00  4.00  5.00  6.00  7.00  8.00
1   1 | 1.00  0.30  1.30  2.30  3.30  4.30  5.30  6.30  7.30
2   1 | 2.00  1.30  0.70  1.70  2.70  3.70  4.70  5.70  6.70
3   2 | 3.00  2.20  1.60  1.10  2.10  3.10  4.10  5.10  6.10
4   3 | 4.00  3.10  2.40  1.90  1.50  2.50  3.50  4.50  5.50
5   4 | 5.00  4.00  3.20  2.60  2.20  1.80  2.80  3.80  4.80
6   5 | 6.00  5.00  4.00  3.30  2.80  2.40  2.00  3.00  4.00
7   6 | 7.00  6.00  5.00  4.00  3.40  2.90  2.50  2.10  3.10
```

the sequences used in the alignment (requested by `tst=2`) and, in addition, the matrix D used in the dynamic programming algorithm (see **6.7.2.1**).

6.7.2.4 Consecutive Indel Operations

In the default alignment procedure, deleting (or inserting) a subsequence consisting of l elements (states) would cost l times the elementary deletion (or insertion) cost. It is sometimes desirable that the deletion (or insertion) cost of such an operation is a more general function of l. Extending the basic algorithm to allow for this modification of the distance measure is quite possible but requires, in general, much more computational effort. See Kruskal and Sankoff [1983, pp. 296–299], Waterman [1995, pp. 194–197]. In general, computation time would be proportional to n^3 (n being the (common) sequence length), instead of n^2 for the basic algorithm. However, if the cost function for multiple indels is linear in l, the basic algorithm can easily be extended to remain proportional in n^2. This section is restricted to this special case.

To explain the extended procedure, we follow Waterman [1995, p. 195]. The terminology follows the introduction in 6.7.2.1. The two sequences for alignment are $a = (a_1, \ldots, a_m)$ and $b = (b_1, \ldots, b_n)$. The substitution cost is given by $w(a_i, b_j)$, and the indel cost function for deleting or inserting a subsequence of l states is assumed to be linear:

$$g(l) = \alpha + \beta(l-1) \qquad \alpha, \beta \geq 0$$

$d_{ij} = (a^i, b^j)$ denotes the optimal distance for the subsequences $a^i = (a_1, \ldots, a_i)$ and $b^j = (b_1, \ldots, b_j)$. The basic recurrence relation is then

$$d_{ij} = \min \begin{cases} d_{i-1,j-1} + w(a_i, b_j) \\ \min_{1 \leq l \leq i} \{d_{i-l,j} + g(l)\} \\ \min_{1 \leq k \leq j} \{d_{i,j-k} + g(k)\} \end{cases}$$

The first of these three possibilities amounts to substituting a_i by b_j; the second results from deleting the last l elements in a^i; and the third results from deleting the last k elements in b^j.

In addition to the matrix $D = (d_{ij})$, the algorithm needs two more matrices: $E = (e_{ij})$ and $F = (f_{ij})$; all with dimension $(m+1, n+1)$.

The algorithm is defined as follows. First there is an initialization:[1]

$$\begin{aligned} d_{0,0} &= e_{0,0} = f_{0,0} = 0 \\ d_{i,0} &= g(i),\ e_{i,0} = g(i) + \alpha \quad \text{for } i = 1, \ldots, m \\ d_{0,j} &= g(j),\ f_{0,j} = g(j) + \alpha \quad \text{for } j = 1, \ldots, n \end{aligned}$$

The recurrence relation is

$$\begin{aligned} e_{ij} &= \min\{d_{i,j-1} + \alpha, e_{i,j-1} + \beta\} \\ f_{ij} &= \min\{d_{i-1,j} + \alpha, f_{i-1,j} + \beta\} \\ d_{ij} &= \min\{d_{i-1,j-1} + w(a_i, b_j),\ e_{ij},\ f_{ij}\} \end{aligned}$$

To see that the algorithm works correctly, one needs to show that

$$\begin{aligned} e_{ij} &= \min_{1 \le k \le j} \{d_{i,j-k} + g(k)\} \\ f_{ij} &= \min_{1 \le l \le i} \{d_{i-l,j} + g(l)\} \end{aligned}$$

The proof is by induction over j for the first equation, and over i for the second equation. To illustrate with the first equation, it certainly holds for $j = 1$. Then, for an arbitrary $j > 1$:

$$\begin{aligned} &\min_{1 \le k \le j} \{d_{i,j-k} + g(k)\} = \\ &\quad \min\{d_{i,j-1} + g(1), \min_{2 \le k \le j} \{d_{i,j-k} + g(k)\}\} = \\ &\quad \min\{d_{i,j-1} + g(1), \min_{2 \le k-1 \le j} \{d_{i,(j-1)-(k-1)} + g(k-1) + \beta\}\} = \\ &\quad \min\{d_{i,j-1} + g(1), \min_{1 \le k \le j-1} \{d_{i,(j-1)-k} + g(k) + \beta\}\} = \\ &\quad \min\{d_{i,j-1} + \alpha, e_{i,j-1} + \beta\} = e_{ij} \end{aligned}$$

In the same way, one can show the second equation.

Syntax. As explained in 6.7.2.2, this extension of the basic distance measure can be selected by specifying an indel cost function with the parameter

`icost =` α, β`,`

where $\alpha \ge 0$ and $\beta \ge 0$.

[1] This is slightly different from Waterman's formulation ([1995, p. 195], but necessary for general α and β.

Box 1 Sequence data file `seqm.d4`

```
Sequence 1     0 0 0 0 0 0
         2     0 1 0 0 0 0
         3     0 1 1 0 0 0
         4     0 1 1 1 0 0
         5     0 1 1 1 1 0
```

Box 2 Command file `seqm4.cf`

```
nvar(
    dfile = seqm.d4,
    Y{1,6} = c1,
);
seqdef = Y1,,Y6;
seqm(
    icost = 1,0.5,
    df = seqm4.tst,
    tst = 2,3,
    dtda = seqm4.tda,
) = seqm4.d;
```

Box 3 Illustration of results for different indel cost functions

```
   default          icost = 1,0.5       icost = 1,0
--------------     --------------     --------------
2 | 2              2 | 2              2 | 2
3 | 4  2           3 | 3  2           3 | 2  2
4 | 6  4  2        4 | 4  3  2        4 | 2  2  2
5 | 8  6  4  2     5 | 5  4  3  2     5 | 2  2  2  2
  ------------       ------------       ------------
    1  2  3  4         1  2  3  4         1  2  3  4
```

Example 1 To illustrate linear indel cost functions, we use the example data file `seqm.d4` in Box 1. The command file is shown in Box 2. Box 3 shows the calculated distances for three different indel cost functions.

6.7.2.5 Data-based Substitution Costs

The distance measure for sequences heavily depends on the specification of substitution and indel cost and, in general, there are only few guidelines. One finally has to find a specification of cost functions that is appropriate for the specific application. One should therefore investigate the behavior of different specifications at least for a subset of "typical" or easily interpretable sequences.

As a starting point, one can use the default cost functions where indel cost = 1, and substitution cost = 2. The alignment of two sequences provides then the longest common subsequence and, as mentioned in **6.7.2.3**, there is a simple relationship between the distance and the length of the longest common subsequence. This option is an interesting starting point because it makes a substitution equivalent to two indel operations (a deletion followed by an insertion), both cost the same.

This shows that the alignment will depend, in particular, on the relation between indel and substitution cost. If substitution cost is higher than the cost of two indel operations, the algorithm will never chose substitutions. It has been suggested, therefore, to set indel cost equal to, or slightly higher, than substitution cost, see Abbott [1990, p. 155]. But as a consequence, the algorithm will then primarily use substitutions and might not be able to fully exploit the possibilities of indel operations.

Further complications arise if the set of elementary operations is extended to include consecutive indel operations, see **6.7.2.4**. In this case, the cost of inserting or deleting l consecutive elements is $g(l) = \alpha + \beta(l-1)$ and one has to find reasonable values for α and β. For some sociological applications, it might be a good idea to set α equal to the highest substitution cost and $0 < \beta < \alpha$.

For some applications, it might also be worthwhile to derive substitution cost from the frequency of transitions in the given sequence data. To explain this idea, let us assume a sample of $i = 1, \ldots, n$ sequences

$$y_i = (y_{i1}, \ldots, y_{iT})$$

where the elements are valid states in the state space $\mathcal{Y} = \{1, \ldots, q\}$, or are (negative) missing values.

We can then define:

$$\begin{aligned} N_t(x) &= \#\text{ of sequences being in state } x \text{ at } t \\ N_{t,t'}(x,y) &= \#\text{ of sequences being in state } x \text{ at } t \\ &\quad \text{and in state } y \text{ at } t' \end{aligned}$$

Using these quantities, there are several possibilities to define substitution cost. The intuition is that two states are less different when there are, in the given data set, less transitions from one state into the other.

Time-independent substitution cost. The most simple way of using the information about transitions would be to ignore the time structure and, using

$$p(x,y) = \frac{\sum_{t=1}^{T-1} N_{t,t+1}(x,y)}{\sum_{t=1}^{T-1} N_t(x)}$$

define substitution cost by

$$w(a_i,b_j) = \begin{cases} 2 - p(a_i,b_j) - p(b_j,a_i) & \text{if } a_i \neq b_j \\ 0 \text{ if } a_i = b_j \end{cases}$$

Then $0 \leq w(a_i,b_j) \leq 2$, and substitution cost directly reflects the cumulated transitions across states. If there are many transitions from a_i to b_j (both directions), substitution cost is low, and vice versa.

This definition of substitution cost can be requested with the parameter `scost=2`; it is easily calculated and does not require additional storage space. If requested, a test output file will show the corresponding substitution cost matrix which, in turn, can be used to find an appropriate indel cost function.

Example 1 To illustrate this option, we use data file `seqm.d4` that was introduced in 6.7.2.3. There are two states, 0 and 1; $p(0,1) = 4/15$ and $p(1,0) = 4/10$. Therefore, $w(0,1) = w(1,0) = 4/3$. See command file `seqm5.cf` in the TDA example archive.

6.7.2.6 Working with Large Samples

Pairwise comparison of n sequences requires $n(n-1)/2$ alignments. If n is big, say greater than 1000, the computational burden becomes prohibitive, in particular with long sequences, since computing time is approximately proportional to the square of the sequence length.

Consider, for example, $n = 1000$ sequences with length 100 and 10 different states. On a SparcStation 10/30, using the current implementation of the alignment algorithm, 100 alignments need about 4 seconds. In this example, $n(n-1)/2 = 499500$, and the total time to calculate these distances would be about 5.5 hours. For $n = 2000$, computing time would already be more than 22 hours. However, in many social science applications, the number of sequences (cases) is several thousand and the calculation of all pairwise distances is no longer feasible. We therefore shortly discuss three possible ways to reduce the computational burden if the number of sequences is large.

Random selection. The easiest way is to select a random subsample and calculate all pairwise distances for the sequences in the subsample. Since this is easily done by a random select statement when reading the input data, a separate option in the `seqm` command is not necessary.

It could be preferable, however, to randomly select pairs of sequences from the full sample. To provide this option, the `seqm` command offers the `r` parameter (only with `m=1`). The syntax is `r=m`, where m is the number of desired alignments. The algorithm to randomly select approximately m pairs of sequences is shown in Box 1.

Predefined sequences for comparison. It sometimes may make sense to compare all sequences with only a few predefined sequences. This might be sensible if one already has some ideas about "typical" sequences to be used for comparison. The `seqm` command provides for this strategy by its `cn` parameter to select a set of sequences for comparison.

Alternatively, one can try a two-step procedure. In a first step, find two sequences having a large distance based on randomly selected pairs of sequences. This can be done since as part of its standard output, TDA prints the case numbers of those two sequences which have the largest distance. Then, in a second step, compare all sequences with these two sequences. The resulting distances can then be plotted in a

Box 1 Random selection of pairs of sequences

```
for (i = 2,...,n) {
    for (j = 1,...,i - 1) {
        r = random number between 0 and 1
        if (r < m / (n (n - 1) / 2))
            calculate distance between sequences i and j;
        else
            skip;
    }
}
```

two-dimensional coordinate system.

Restricted alignment. Another strategy to reduce the computational burden is to restrict the possible alignments of sequences, see Kruskal and Sankoff [1983, pp. 276–281]. Given two sequences, $a = (a_1, \ldots, a_m)$ and $b = (b_1, \ldots, b_n)$, the algorithm described in **6.7.2.1** requires the recursive calculation of an $(m+1, n+1)$ matrix $D = (d_{ij})$; d_{ij} being the optimal distance between the subsequences a^i and b^j.

The idea is to calculate only a part of this D matrix, for instance, only elements with $|\, i - j \,| \leq K$. In effect, this excludes some possible alignments from consideration; but this might be justifiable not only for computational reasons but also with respect to the application. As discussed by Kruskal and Sankoff [1983], p. 278], a suitable formula that covers the case of unequal sequence lengths is

$$\left| \frac{i}{m+1} - \frac{j}{n+1} \right| \leq K \frac{1}{\sqrt{2}} \left[\frac{m+1}{n+1} + \frac{n+1}{m+1} \right] \Big/ \sqrt{(m+1)^2 + (n+1)^2}$$

(In the special case that $m = n$, this reduces to $|\, i - j \,| \leq K$.) The `seqm` command provides this option with the `max` parameter. The syntax is `max=`K with an integer $K \geq 1$. Then only that part of the D matrix is calculated and used to find an optimal alignment where the indices are in the range given by the above mentioned formula. For a variety of other approaches to restrict the alignment of sequences, see the discussion in Kruskal and Sankoff [1983].

6.7.2.7 Comparing Parallel Sequences

Sometimes one is interested in comparing pairs of sequences where each pair of sequences is defined simultaneously for each case in the input data. A typical application would be a comparison of partners living together in some relationship. This can be done with the `m=2` option.

The option requires to specify two sequence data structures with the parameter `sn=`k_1, k_2. There are then two sequences for each case in the input data to be compared and the result will be written into the output file. Given n cases, the output file will contain just n records.

A useful additional option is provided by the `s=1` parameter providing information about how the distance between the two sequences evolves over time. This could be useful, for instance, for a dynamical investigation of homogamy.

6.9 Least Squares Regression

This chapter deals with least squares regression. Available commands are described in the following subsections.

6.9.1 The `lsreg` Command to be used for linear OLS regression.

6.9.2 Regression with Censored Data describes the `lsreg1` command that can be used when observations of the dependent variable are right censored. Note that this is currently only an experimental command.

6.9.1 The `lsreg` Command

The discussion of least squares estimation in this section is confined to cross-section data. There is a single dependent variable, Y_i, and a vector of exogenous (non-stochastic) covariates, X_i, for cases $i = 1, \ldots, n$. For ease of notation, X_i is taken as a row vector including a constant one to provide an intercept term in the regression model. As an option one can suppress the intercept, see below.

An ordinary least squares (OLS) model can be written as

$$Y_i = X_i\beta + \epsilon_i \qquad i = 1, \ldots, n$$

β is the parameter vector with length equal to the number of covariates in X_i. ϵ_i is the residual for individual i. The same model may be written in matrix notation as

$$Y = X\beta + \epsilon \tag{1}$$

With m covariates, X is an (n, m) matrix, Y and ϵ are both vectors of length n, and β is a vector of length m.

A least squares estimate of the parameter vector β is found by minimizing the Euclidean norm of the residuals as a function of the model parameters, i.e.

$$\| \epsilon \| \quad = \quad \left(\sum_{i=1}^{n} \epsilon_i^2 \right)^{1/2} \rightarrow \min \tag{2}$$

The solution, if existent, is found by

$$\hat{\beta} = (X'X)^{-1} X'Y \tag{3}$$

where X' is the transpose of X. A unique solution exists if X has full column rank; this implies that $n \geq m$.

Several different algorithms can be used to solve (2). TDA's main algorithm to solve least squares problems is based on algorithms published by Hanson and Haskell [1982]. These algorithms extend earlier work of Lawson and Hanson [1974], providing, in particular, the possibility to solve the least squares problem with linear equality and inequality constraints. A variety of other algorithms is discussed in Späth [1992].

Box 1 Syntax for `lsreg` command

```
lsreg (
      ni=1,                 if without intercept, def. ni=0
      lsecon=...,           equality constraints, see 6.9.1.2
      lsicon=...,           inequality constraints, see 6.9.1.2
      dgrp=...,             estimate dummy variables with constraints
      tfmt=...,             print format for results, def. tfmt=10.4
      s=1,                  use robust covariance matrix, def. s=0
      df=...,               print data to an output file
      fmt=...,              print format for df option, def. 10.4
      dtda=...,             TDA description file for df option
      ppar=...,             print estimated coefficients to output file
      pcov=...,             print covariance matrix to output file
      pres=...,             print residuals to output file
      mfmt=...,             print format for pcov and pres option
      mplog=...,            write norm of residuals into matrix
      mppar=...,            write estimated parameters into matrix
      mpcov=...,            write estimated covariance matrix
                            into matrix
) = varlist;
```

Command Description. The command for least squares regression is `lsreg`. Its syntax is

```
lsreg (parameter) = varlist;
```

The optional parameters are described below, a summary is given in Box 1. The right-hand side must provide a list of variables for the regression. It is a comma-separated list of variable names that must be given in the following form:

$$Y_1, X_1, X_2, \ldots, X_m$$

First comes the name of the dependent variable, then follow names of independent variables. There must be at least one variable name.

1. As default, the model specification contains an intercept term. Using the parameter `ni=1` estimates a model without an intercept.

2. The parameters `lsecon` and `lsicon` can be used to specify parameter constraints and will be explained in 6.9.1.2.

3. The `dgrp` parameter can be used to specify sets of dummy variables to be estimated as deviations from a weighted mean, see **6.9.1.3**.

4. Results of model estimation are written into the standard output. The print format can be controlled with the `tfmt` parameter, default is `tfmt=10.4`. A description of the output is given below in Example 1.

5. The data used for model estimation can be written into an output file with the parameter

```
df = name_of_an_output_file,
```

If there are n cases and m independent variables, the output file will contain n records. Each record has $m + 1$ entries, first the independent variables and finally the dependent variable. The print format can be controlled with the `fmt` parameter, default is `fmt=10.4`. In connection with the `df` parameter one can request an additional output file with the `dtda` parameter containing a TDA description of the data file.

6. The parameter

```
ppar = name_of_an_output_file,
```

can be used to write the estimated model coefficients (and standard errors) into an output file. The print format is determined by `tfmt`.

7. As default, calculation of standard errors uses the least squares covariance matrix

$$\hat{\sigma}^2 \, (X'X)^{-1}$$

where $\hat{\sigma}^2$ is the estimated variance of residuals. Optionally, one can use the parameter `s=1` to request a "robust" covariance matrix as proposed by White [1980]. The covariance matrix is then calculated as

$$(X'X)^{-1} \, (X' \, \mathrm{diag}\{e_i^2\} \, X) \, (X'X)^{-1}$$

where $\mathrm{diag}\{e_i^2\}$ is an (n, n) diagonal matrix containing the squared residuals.[1]

[1] There are also other proposals for defining a robust covariance matrix, see Greene [1992, p. 250]. The TDA example archive contains a command file, `lsreg12.cf`, that can be used to replicate the example given there. The command file `lsreg12a.cf` contains a TDA matrix procedure for calculating White's robust covariance matrix.

Box 2 Command file `lsreg1.cf`

```
nvar(
    dfile = lsreg1.dat,      # data file
    Height = c1,
    Weight = c2,
);
lsreg(
    pcov = cov,              # write covariance matrix to cov
    ppar = par,              # write parameter to par
    df = df,                 # write data to df
    pres = res,              # write residuals to res

) = Weight,Height;           # variables
```

8. The parameter

```
pcov = name_of_an_output_file,
```

can be used to write the estimated covariance matrix into an output file. This will depend on the `s` parameter. The print format can be controlled with the `mfmt` parameter, default is `mfmt=12.4`.

9. The `pres` parameter can be used to request an additional output file with regression residuals, see 6.9.1.1.

Example 1 To illustrate the output of the `lsreg` command, we use a simple example from SAS/STAT User's Guide, Vol. 2, p. 1354. The data file, `lsreg1.dat`, contains two variables, `Weight`, and `Height`. Box 2 shows the command file, `lsreg1.cf`.

The `nvar` command reads the data file and defines the two variables, `Weight` and `Height`. Then follows the `lsreg` command for least squares regression. `Weight` is the dependent and `Height` is the independent variable. The command contains parameters to write for additional output files. Standard output from the `lsreg` command is shown in Box 3. It begins with information about the variables and number of cases. Then follows some information about the least squares solution.

1. The rank of the least squares data matrix is the rank of the matrix X in (1). For a full rank situation, it should be equal to the number of independent variables, including (if not suppressed) the intercept. If the data matrix is rank deficient, TDA's least squares algorithm calculates a solution based on a generalized inverse. This solution is not unique. A warning message is then written into the standard output.

Box 3 Part of standard output from command file `lsreg1.cf`

```
Least squares regression.

Variables (cross-section)
-------------------------
Y   : Weight
X1  : Height

Equality constraints: 0
Inequality constraints: 0

Reading data. Cases: 19
Data written to: df

Rank of least squares data matrix: 2
Norm of least squares residuals: 46.287
Degrees of freedom: 17
Sum of squared residuals: 2142.49
Variance of residuals: 126.029
Squared multiple correlation: 0.770507
Adjusted: 0.757007
F-statistic: 57.0763
Level of significance: 0.999999
Covariance matrix written to: cov

Idx  Wave  Variable       Coeff      Error    Coeff/E  Signif
----------------------------------------------------------------
  1     -  Intercept  -143.0269    32.2746    -4.4316  0.9996
  2     1  Height        3.8990     0.5161     7.5549  1.0000

Parameter estimates written to: par
Residuals written to: res
```

2. Then follows the norm of the least squares residuals, see (2).

3. The degrees of freedom is the number of rows minus the number of linear independent columns in the data matrix X.

4. The sum of squared residuals, s_r, is calculated as the square of the norm of the residuals.

5. The variance of the residuals is s_r/d, that is, the sum of squared residuals divided by the number of degrees of freedom.

6. The squared multiple correlation is calculated as

$$R^2 = \frac{s_q - s_r}{s_q} \quad \text{where } s_q = s_{yy} - \frac{s_y s_y}{n},\ s_y = \Sigma y_i,\ s_{yy} = \Sigma y_i^2$$

y_i $(i = 1, \ldots, n)$ are the values of the dependent variable. The adjusted R^2 is only calculated if the number of degrees of freedom, d, is positive. The formula is

$$R^2_{\text{adj}} = 1 - (n-1)(1 - R^2) \,/\, d$$

7. The value of the F-statistic is calculated as

$$F = \frac{R^2/(m-1)}{(1-R^2)/d}$$

given that the data matrix has full column rank and there is a positive number of degrees of freedom. m is the number of independent variables, including the intercept. The significance level is the value of the F-distribution, at F, with $m-1$ and d degrees of freedom.

8. Finally, the `lsreg` command prints a table with estimated coefficients, $\hat{\beta}_j$, and standard errors, $\hat{\sigma}_j$. If the model is specified without constraints, standard errors are only calculated if the data matrix has full column rank and there is a positive number of degrees of freedom. The significance is calculated as $2\,t(\hat{\beta}_j/\hat{\sigma}_j, d) - 1$ from the t-distribution with d degrees of freedom. As described above, standard errors depend on the type of covariance matrix. As default, TDA uses the standard OLS covariance matrix. If using the `s=1` parameter, calculation of standard errors is based on White's [1980] robust covariance matrix.

6.9.1.1 Residuals

When using the `lsreg` command for least squares regression, the `pres` parameter can be used to request an additional output file containing regression residuals. The syntax is

```
pres = name_of_an_output_file,
```

The number of records in the output file equals the number of data matrix cases used for the regression. Each record of the output file contains the following entries.

1. The case number.
2. Values of the independent variables $X_{i1}, \ldots, X_{im}$ used in the regression model, including (if not suppressed) a constant 1 for the intercept.
3. The value of the dependent variable, Y_i.
4. The estimate of the dependent variable, $\hat{Y}_i$.
5. The residual, $e_i = Y_i - \hat{Y}_i$.
6. The leverage value, h_{ii}, i.e., the diagonal elements in $X(X'X)^{-1}X'$.
7. An estimate for the standard deviation of the residual, $\mathrm{SD}(e_i)$, calculated as

   $$\mathrm{SD}(e_i) \;=\; \hat{\sigma}\,\sqrt{1 - h_{ii}}$$

   $\hat{\sigma}$ is the estimated standard deviation of residuals.
8. The standardized residuals, $e_i / \mathrm{SD}(e_i)$.

The output file can be used as a data input file in subsequent calls of TDA to investigate and plot residuals; some header lines describe the variables. The print format can be controlled with the `mfmt` parameter, default is `mfmt=12.4`. Leverage values and standard deviations for residuals are only calculated if there is a valid estimate for the covariance matrix, otherwise the corresponding columns in the output file will contain the value -1.

6.9.1.2 Parameter Constraints

The least squares problem described in 6.9.1 can be modified by imposing constraints for the model parameters, often called restricted least squares. Our discussion will be confined to two kinds of linear constraints. First, equality constraints which may be written

$$R\beta = d \tag{1}$$

where R is an arbitrary (n_r, m) matrix and d is a vector of length n_r. Second, there may be inequality constraints which can be written

$$U\beta \geq h$$

with U an arbitrary (n_u, m) matrix and h a vector of length n_u. In both cases, the constraints are assumed to be non-stochastic.

Taking into account the possibility of linear equality and/or inequality constraints, a general least squares problem may be written as

$$\begin{bmatrix} R \\ X \\ U \end{bmatrix} \beta \doteq \begin{bmatrix} d \\ Y \\ h \end{bmatrix} \begin{matrix} \} & n_r \\ \} & n_x \\ \} & n_u \end{matrix} \tag{2}$$

n_r is the number of equations to be satisfied exactly, n_x (equal to n as used in the introductory section) is the number of equations to be satisfied in a least squares sense, and n_u is the number of inequality constraints. The symbol $\doteq$ is used to denote an equality, a least squares, and an inequality relation, respectively. Some of the many possible combinations are shortly discussed.

1. If $n_x > 0$ and $n_r = n_u = 0$, the general system (2) reduces to a standard regression approach without constraints. TDA tries to calculate the standard OLS solution as defined in 6.9.1. A unique solution exists if $n_x \geq m$ and X has full column rank, m. If X has less than full column rank, many solutions exist that equally well minimize the norm of the residuals. TDA gives one of these solutions. In any case, the program provides information about the rank of X. Due to limited numerical accuracy, this is actually a pseudo rank.

2. If $n_r > 0$ and $n_x = n_u = 0$, one has a simple system of linear equations. If $n_r = m$ and R has full column rank, m, there is exactly one solution. If $n_r < m$ there may be a unique solution, or a set of different solutions, or the equations may be contradictory. If the solution is not unique, TDA provides just one. In any case, one gets information about the number of independent rows in R and about whether the system is contradictory or not. If $n_r > m$ the system is solved in a least squares sense.

3. If $n_u > 0$ and $n_x = n_r = 0$, one has a pure system of inequalities. Again, there may be a unique solution (unique in the sense of a solution vector with minimal Euclidean length), there may exist many (linear independent) solutions, or the inequalities may be contradictory. If possible, TDA calculates a solution with minimal norm.

4. If $n_x > 0$, $n_r > 0$, and $n_u = 0$, one has a least squares problem with equality constraints. To find a solution of this problem is tried in two steps. First, it is tried to solve the equality constraints. If there is no, or exactly one, solution, the least squares equations are ignored. If the solution space has a positive dimension, the least squares problem is reformulated in this solution space and an optimal solution, based on the least squares equations, is determined. Of course, a unique solution is not guaranteed if X is rank deficient.

5. If $n_x > 0$, $n_u > 0$, and $n_r = 0$, one has a least squares problem with inequality constraints. To find an optimal solution the system of inequalities must not be contradictory. In addition, the rank of the least squares equations must be at least equal to the dimension of the solution space of the inequality constraints.

6. Finally, if $n_x > 0$, $n_r > 0$, and $n_u > 0$, one has a least squares problem with both, equality and inequality constraints. Again, it is tried to find a solution by first solving the system of equality constraints. The remaining problem is reformulated, then, in the solution space of the equality system. If this solution space has positive dimension, the algorithm proceeds in the same way as explained above.

Specification of constraints. TDA uses all rows of the data matrix to build the least squares problem, $X\beta \cong Y$; constraints are only added if defined explicitly. Following (2), the syntax to define equality constraints is

$$\text{lsecon} \;=\; r_{i0}\,\mathrm{b}_0 \pm r_{i1}\,\mathrm{b}_1 \pm \ldots \pm r_{in}\,\mathrm{b}_n \;=\; d_i \qquad (3)$$

r_{ij} and d_i are floating point numbers to be used as the coefficients of the matrix R and the vector d, respectively. The b_j are key words and must be given as `b0,b1,...` There must be a separate `lsecon` parameter for each equality constraint. If r_{ij} is zero, the term can be omitted. `b0` can only be used if the model contains an intercept term.

Analogously, the syntax for the specification of inequality constraints is

$$\text{lsicon} \;=\; u_{i0}\,\mathrm{b}_0 \pm u_{i1}\,\mathrm{b}_1 \pm \ldots \pm u_{in}\,\mathrm{b}_n \;=\; h_i \tag{4}$$

with u_{ij} and h_i, respectively, the coefficients of the U matrix and the h vector, see (2).

Covariance matrix. Given a regression model with n_r equality constraints,[1] the least squares estimator is the solution of

$$(Y - X\beta)'(Y - X\beta) \rightarrow \min \qquad \text{s.t.} \quad R\beta = d$$

The solution can be found by using an n_r vector, λ, of Lagrangian multipliers, i.e. by minimizing

$$L(\beta, \lambda) = (Y - X\beta)'(Y - X\beta) + 2\,\lambda'(R\beta - d)$$

In matrix notation, the first-order conditions can be written

$$\begin{bmatrix} X'X & R' \\ R & 0 \end{bmatrix} \begin{bmatrix} \beta \\ \lambda \end{bmatrix} \;=\; \begin{bmatrix} X'Y \\ d \end{bmatrix} \tag{5}$$

We shall assume that there is a unique solution, $(\hat{\beta}, \hat{\lambda})'$, i.e. that the matrix on the left-hand side has full rank, $m + n_r$. This implies that the equality constraints are linearly independent. As pointed out by Greene and Seaks [1991], it is not required that also X has full rank. As an implication of the first-order conditions, one finds that

$$X'X(\hat{\beta} - \beta) = X'\epsilon - R'\hat{\lambda}$$

This shows that, if the unrestricted least squares estimator exists (X has full rank), then $\hat{\beta}$ is an unbiased estimator of β,

$$\hat{\beta} - \beta = (X'X)^{-1}X'\epsilon - (X'X)^{-1}R'\hat{\lambda}$$

[1] A covariance matrix is only calculated if there are no inequality constraints. Derivation of stochastic properties for models with both, equality and inequality constraints is difficult; see, for instance, Judge et al. [1988, chap. 20]. So we shall currently not discuss the case of inequality constraints.

Box 1 Data file `lsreg2.dat`

```
 X1  X2   Y
------------
  1   0   1
  0   1   1
  1   1   1
```

Box 2 Command file `lsreg2.cf`

```
nvar(
    dfile = lsreg2.dat,
    X1 = c1,
    X2 = c2,
    Y  = c3,
);
lsreg (
    ni=1,           # without intercept
    lsecon = b1 + b2 = 6,
    lsicon = b1 = 2,
    lsicon = b2 = 2,
) = Y,X1,X2;        # variables
```

and the covariance matrix of $\hat{\beta}$ may be derived[2] as

$$\mathrm{Var}[\hat{\beta}] = \sigma^2 \left[I - (X'X)^{-1}R' \left[R(X'X)^{-1}R' \right]^{-1} R \right] (X'X)^{-1}$$

As shown by Greene and Seaks [1992], there is a more general formulation of the covariance matrix of $\hat{\beta}$ which does not require that the unrestricted least squares estimator exists. Using W to denote the matrix on the left-hand side of (5), it follows that

$$\begin{aligned} \mathrm{Var} \begin{bmatrix} \hat{\beta} \\ \hat{\lambda} \end{bmatrix} &= W^{-1} \mathrm{Var} \begin{bmatrix} X'Y \\ d \end{bmatrix} W^{-1} \\ &= W^{-1} \begin{bmatrix} \sigma^2 X'X & 0 \\ 0 & 0 \end{bmatrix} W^{-1} \end{aligned}$$

It is only required that W has full rank. The formula may be further simplified as shown by Greene and Seaks [1991].

[2] See, e.g., Johnston [1972, p. 158].

Box 3 Part of standard output from command file `lsreg2.cf`

```
Least squares regression. Current memory: 144677 bytes.

Variables (cross-section)
-------------------------
Y   : Y
X1  : X1
X2  : X2

Model without intercept.

LSECon: 1 b1 + 1 b2 = 6
LSICon: 1 b1 = 2
LSICon: 1 b2 = 2

Equality constraints: 1
Inequality constraints: 2

Reading data. Cases: 3

Covariance matrix not calculated.
Rank of reduced least squares data matrix: 1
Norm of least squares residuals: 5.74456
Rank of equality constraints: 1
Norm of residuals of equality constraints: 0

Sum of squared residuals: 33

Idx  Wave  Variable       Coeff      Error    Coeff/E  Signif
-----------------------------------------------------------------
  1     1  X1            3.0000        ---        ---     ---
  2     1  X2            3.0000        ---        ---     ---
```

Example 1 To illustrate constrained least squares we consider the following example:

$$\begin{array}{rcrcl} \beta_1 & + & \beta_2 & = & 6 \\ \beta_1 & & & \cong & 1 \\ & & \beta_2 & \cong & 1 \\ \beta_1 & + & \beta_2 & \cong & 1 \\ \beta_1 & & & \geq & 2 \\ & & \beta_2 & \geq & 2 \end{array}$$

The data correspond to example 51 in Späth [1992, p. 322]. In this example, $n_r = 1$, $n_x = 3$, and $n_u = 2$. Box 2 shows the command file

`lsreg2.cf` that can be used to find a solution, Box 3 shows part of the standard output. Since the rank of the reduced least squares matrix is less than 2, there are no standard errors.

6.9.1.3 Dummy Variables

Let $D_1, \ldots, D_k$ denote a complete set of dummy variables, meaning that for each case in the data set exactly one of these variables takes the value 1. Since the intercept column is linear dependent on this set of dummy variables, one normally has to drop one of them that creates the reference category. Using parameter constraints provides an alternative. One can keep the full set of dummy variables and add the constraint

$$\sum_j w_j D_j \beta_j = 0$$

with w_i some set of weights; $w_i > 0$, $\Sigma_i w_i = 1$, and β_j the regression parameters corresponding to the dummy variables D_j. This can be specified with the `lsecon` parameter.

As discussed by Haisken-DeNew and Schmidt [1997], a reasonable approach for many applications is to define the weights to be the proportion of cases in the respective category. This is supported by the `dgrp` parameter in the `lsreg` command. The syntax is

```
dgrp = [DL1],[DL2],...,
```

and `DL1`, `DL2`, and so on are namelists containing dummy variables which are part of the set of independent variables used in the currently specified regression model. For each of these namelists TDA calculates the proportion of cases in each dummy variable and creates an equality constraint using these proportions as weights. Up to 20 sets of dummy variables can be specified on the right-hand side of the `dgrp` parameter. For example, assume that the `lsreg` command specifies the variables

`lsreg (parameter) =` $Y, D_1, D_2, D_3, D_4, U_1, U_2, U_3, X_1, \ldots;$

where `DL1 =` $\{D_1, D_2, D_3, D_4\}$ and `DL2 =` $\{U_1, U_2, U_3\}$ are two sets of dummy variables. Given then the parameter

```
dgrp = [DL1],[DL2],
```

TDA creates two equality constraints:

$$d_1 D_1 \beta_1 + d_2 D_2 \beta_2 + d_3 D_3 \beta_3 + d_4 D_4 \beta_4 = 0$$

Box 1 Command file `lsreg3.cf`

```
nvar(
    dfile = lsreg3.dat,
    G  = c2,
    Y  = c3,
    NE = c4,
    NC = c5,
    SO = c6,
    WE = c7,
    Y2 = Y * Y,
);
nlist(
    DL = NE,NC,SO,WE,
);
lsreg (
    dgrp = [DL],
    tfmt = -14.6,
) = G,Y,Y2,DL;
```

and

$$u_1 U_1 \gamma_1 + u_2 U_2 \gamma_2 + u_3 U_3 \gamma_3 = 0$$

with d_i and u_i the proportion of cases in the corresponding category. The regression model is then estimated by using these constraints (in addition to any constraints explicitly defined with the `lsecon` and/or `lsicon` parameters).

Information about the dummy variables and the corresponding proportion of cases is written into the standard output. In addition, as proposed by Haisken-DeNew and Schmidt [1997], TDA calculates, separately for each group of dummy variables, a dispersion measure. If $\mathcal{D}_i$ denotes the index set for the ith set of dummy variables, this dispersion measure is calculated as

$$\sqrt{\sum_{j \in \mathcal{D}_i} w_j \left(\hat{\beta}_j^2 - \hat{\sigma}_j^2\right)}$$

where w_j are the weights of the dummy variables and $\hat{\beta}_j$ and $\hat{\sigma}_j$ are the corresponding parameter and standard error estimates.

Example 1 To illustrate the `dgrp` parameter, we replicate an example given in Haisken-DeNew and Schmidt [1997]. The data file, `lsreg3.dat`, contains information about gasoline demand in four different regions

Box 2 Standard output from `lsreg3.cf`

```
Least squares regression. Current memory: 268781 bytes.

Processing dgrp option.
Group selection: [DL]

Variable     cases       weight
------------------------------
NE              12       0.2500
NC              12       0.2500
SO              12       0.2500
WE              12       0.2500

Equality constraints: 1
Inequality constraints: 0

Reading data. Cases: 48

Rank of reduced least squares data matrix: 6
Norm of least squares residuals: 3.50111
Rank of equality constraints: 1
Norm of residuals of equality constraints: 0

Degrees of freedom: 42
Sum of squared residuals: 12.2578
Variance of residuals: 0.291852
Squared multiple correlation: 0.973878
Adjusted: 0.970768

Idx  Wave  Variable          Coeff          Error        Coeff/E
-------------------------------------------------------------------
  1     -  Intercept   1.045236e+00   1.978744e-01   5.282321e+00
  2     1  Y           7.778210e-04   2.982920e-05   2.607583e+01
  3     1  Y2         -1.332484e-08   7.953897e-10  -1.675260e+01
  4     1  NE         -7.873645e-01   1.351198e-01  -5.827160e+00
  5     1  NC         -3.929084e-02   1.351144e-01  -2.907968e-01
  6     1  SO          4.296827e-01   1.351670e-01   3.178902e+00
  7     1  WE          3.969727e-01   1.350824e-01   2.938745e+00

DGroup  weighted adj. stand. dev.
---------------------------------
[DL]                4.718767e-01
```

and is taken from Suits [1984].[1] The variables are `G` (gasoline demand), `Y` (income), and four dummy variables for regions (`NE,NC,SO,WE`).

Box 1 shows the command file `lsreg3.cf`. The `lsreg` command specifies a simple regression model with dependent variable `G`. The independent variables are `Y`, `Y` squared, and the four dummy variables. As requested by the `dgrp` option, these dummy variables are treated as a complete set and a corresponding constraint is added to the model specification.

Box 2 shows the standard output. In this example, there are 12 cases in each region, so all weights are equal to 0.25 and the equality constraint is simply

$$0.25\,\texttt{NE} + 0.25\,\texttt{NC} + 0.25\,\texttt{SO} + 0.25\,\texttt{WE} = 0$$

[1] The same data were used by Greene and Seaks [1991].

6.9.2 Regression with Censored Data

This section describes the `lsreg1` command that can be used for least squares estimation when dependent variables refer to possibly right censored observations. Note that the current implementation is experimental and not yet finished.

The command is intended to estimate parameters for several regression equations simultaneously. The basic model consists of n regression equations

$$\begin{aligned} Y_1 &= x_1\beta_1 + \epsilon_1 \\ Y_2 &= x_2\beta_2 + \epsilon_2 \\ &\vdots \\ Y_n &= x_n\beta_n + \epsilon_n \end{aligned}$$

Y_j is the dependent (random) variable in the jth regression equation, x_j is a $(1, m)$ vector of given covariates, and ϵ_j is the corresponding residual. It is assumed that, for each dimension $j = 1, \ldots, n$, observations are given in the following way:

$$\begin{array}{cccc} y_{j1} & \delta_{j1} & x_{j,11} & \cdots & x_{j,1m} \\ y_{j2} & \delta_{j2} & x_{j,21} & \cdots & x_{j.2m} \\ \vdots & \vdots & \vdots & & \vdots \\ y_{jn_j} & \delta_{jn_j} & x_{j,n_j1} & \cdots & x_{j,n_jm} \end{array}$$

n_j is the number of observations for the jth regression equation (dimension). y_{ji} is the ith observed value for dependent variable Y_j; δ_{ji} is a corresponding censoring indicator: if $\delta_{ji} = 1$, y_{ji} is assumed to be an exactly observed value, if $\delta_{ji} = 0$, y_{ji} is assumed to be a right censored observation.

Box 1 shows the syntax for the `lsreg1` command. Most parameters are optional. Required is the name of the dependent variable (`yl` parameter) and the name of a variable containing the censoring information (`cen` parameter).

1. How the command estimates model parameters depends on the selected option (`opt` parameter). Available options will be explained below.

Box 1 Syntax for `lsreg1` command

```
lsreg1 (
    opt=...,        calculation of expected values, def. 1
                    1 = based on marginal Kaplan-Meier
                    2 = based on joint distribution, method 1
                    3 = based on joint distribution, method 2
    yl=...,         name of dependent variable
    cen=...,        variable name for censoring information
    ni=1,           if without intercept, def. ni=0
    grp=ID,L1,      specification of dimensions
    mxit=...,       maximal number of iterations, def. 20
    tolp=...,       tolerance for convergence, def. 0.0001
    sc=...,         offset for domain (method 1 and 2), def. 0
    n=...,          number of boxes in grid (method 1), def. 100
    mxitl=...,      maximal number of iterations (method 1), def. 20
    tolf=...,       tolerance for convergence (method 1), def. 0.001
    d=...,          delta specification (method 2), def. 0.1
    tfmt=...,       print format for results, def. 10.4
    prot=...,       protocol file for diagnostic information
    ppar=...,       print estimated parameters to output file
    pres=...,       print data and residuals to output file
    fmt=...,        print format for pres option, def. 10.4
    dtda=...,       TDA description file for pres option
) = varlist (indep. variables);
```

2. Independent variables can be specified on the right-hand side. If no variables are given, the command estimates a model where each regression equation only contains a single intercept. By default, an intercept is added to each regression equation. The `ni = 1` parameter can be used to suppress intercepts. Of course, the command then needs at least one independent variable.

3. By default, the command assumes a single regression equation where each row in the current data matrix provides one observation for this regression equation ($n = 1, n_1 =$ NOC). The `grp` parameter can be used to specify two or more regression equations. The syntax is

```
grp = ID, L1,
```

where `ID` and `L1` are names of data matrix variables. Each block of data matrix rows where `ID` has identical values is interpreted as data for one

unit of observation. Any values are possible and since the data matrix is always sorted with respect to `ID` and `L1`, it is not required that blocks are contiguous. The `L1` variable must contain positive integers. The number of different integers found in this variable is interpreted as the number of regression equations (dimensions). Again, it is not required that these numbers are contiguous. Each data matrix row provides one observation for the regression equation specified by the corresponding value of the `L1` variable. (For an illustration, see the description of the `gdf` command in **6.2.2**.)

4. Resulting parameter estimates are written into the standard output, the print format can be specified with the `tfmt` parameter. Parameter estimates are also written to an output file when an output file name is given with the `ppar` parameter.

5. A further output file can be requested with the `pres` parameter. This file will then contain a copy of the data, expected values of the dependent variables, given the data, and residuals. A description of the variables contained in this file can be requested with the `dtda` parameter. The print format can be controlled with the `fmt` parameter.

6. The remaining parameters are used to control the iterative estimation procedure and depend on which option is selected with the `opt` parameter. They will be explained below.

Estimation Procedure

The estimation procedure is a multivariate generalization of an approach originally proposed by Buckley and James [1979]. Assume the data are given by

$$X = \begin{bmatrix} x_{1,11} & \cdots & x_{1,1m} \\ \vdots & & \vdots \\ x_{1,n_1 1} & \cdots & x_{1,n_1 m} \\ \vdots & & \vdots \\ x_{n,11} & \cdots & x_{n,1m} \\ \vdots & & \vdots \\ x_{n,n_n 1} & \cdots & x_{n,n_n m} \end{bmatrix} \quad y = \begin{bmatrix} y_{1,1} \\ \vdots \\ y_{1,n_1} \\ \vdots \\ y_{n,1} \\ \vdots \\ y_{n,n_n} \end{bmatrix} \quad \delta = \begin{bmatrix} \delta_{1,1} \\ \vdots \\ \delta_{1,n_1} \\ \vdots \\ \delta_{n,1} \\ \vdots \\ \delta_{n,n_n} \end{bmatrix}$$

Ignoring censoring, standard OLS estimation would result in

$$\hat{\beta} = (X'X)^{-1}X'y$$

Box 2 Iterative algorithm for parameter estimation

(1) $b = (X'X)^{-1}X'y$

(2) $e = y - Xb$

(3) calculate new values for dependent variable:

$$y^*_{ji} = \begin{cases} y_{ji} & \text{if not censored} \\ \text{conditional expectation, based on data and } e \end{cases}$$

(4) $b^* = (X'X)^{-1}X'y^*$

(5) check convergence: $\max_k \left\{ \frac{|\, b^*_k - b_k \,|}{\max\{b^*_k, 1\}} \right\} \leq \text{TOLP}$

(6) end if convergence has been achieved, or the maximal number of iterations has been reached.

(7) $b = b^*$, continue with step (2)

These parameter estimates are then used as starting values for an iterative procedure where, in each iteration, censored observations are substituted by their conditional expectations, given the data and the current values of the parameter estimates. Box 2 shows the basic steps of the iterative algorithm.

A problem only occurs in step (3) where we have to calculate expected values for the censored observations, conditional on data and current values of model parameters. The proposal made by Buckley and James [1979] was to use

$$y^*_{ji} = \delta_{ji} y_{ji} + (1 - \delta_{ji})\, E_{\hat{F}_j}(Y_j \,|\, Y_j > y_{ji}, x_{ji}, b_j, e_j) \tag{1}$$

where b_j and e_j refer, respectively, to the current parameter estimates and residuals for the jth regression equation, and $\hat{F}_j$ is the Kaplan-Meier estimate of the distribution function of these residuals. For censored observations, the conditional expectation is then calculates by

$$\begin{aligned} & E_{\hat{F}_j}(Y_j \,|\, Y_j > y_{ji}, x_{ji}, b_j, e_j) \\ &= E_{\hat{F}_j}(x_{ji}b_j + \epsilon_j \,|\, y_{ji} - x_{ji}b_j < \epsilon_j \leq y_{ji} - x_{ji}b_j, x_{ji}, b_j) \\ &= x_{ji}b_j + E_{\hat{F}_j}(\epsilon_j \,|\, y_{ji} - x_{ji}b_j < \epsilon_j \leq y_{ji} - x_{ji}b_j, x_{ji}, b_j) \\ &= x_{ji}b_j + \int_{y_{ji} - x_{ji}b_j}^{\infty} e \, d\hat{F}_j \;/\; \int_{y_{ji} - x_{ji}b_j}^{\infty} d\hat{F}_j \end{aligned}$$

This method to calculate conditional expectations for censored observations is implemented as the default option (`opt = 1`) of the `lsreg1` command. The tolerance, TOLP, can be specified with the `tolp` parameter, the maximal number of iterations with the `mxit` parameter.

6.10 Alternative Regression Methods

While least squares regression is most often used in applied work, there are many alternative approaches, see, e.g., Birkes and Dodge [1993]. This chapter is intended to discuss some of these alternatives. Currently, we have the following sections.

6.10.1 L1-Norm Regression

An obvious alternative to least squares regression is to minimize absolute deviations, instead of squared deviations. This is called L_1 norm regression. See, e.g., Dielman and Pfaffenberger [1982], Birkes and Dodge [1993, ch. 4]. For a comprehensive bibliography see Dielman [1984]. Given a sample of data, (y_i, x_i), where x_i is a row vector of covariates, a linear L_1 norm regression will find parameter estimates by minimizing

$$\sum_{i=1}^{n} | y_i - x_i\beta | \longrightarrow \min$$

The TDA command to solve this problem is called `l1reg`, its syntax is shown in Box 1. The command uses a linear programming algorithm developed by Barrodale and Roberts [1974]. For an extensive discussion of this and related algorithms see also Späth [1987]. The syntax for the `l1reg` command is

```
l1reg (parameter) = varlist;
```

The right-hand side must provide a list of variables for the regression. It is a comma-separated list of variable names that must be given in the following form:

$$Y_1, X_1, X_2, \ldots, X_m$$

First comes the name of the dependent variable, then follow names of independent variables. The `l1reg` command can only be used with cross-sectional data.

The solution of the L_1 norm regression problem is not necessarily unique. Information is given in the programm's standard output. It may also happen that the algorithm is not able to find a solution. If successful, the `l1reg` command prints a table with estimated parameters into the standard output.

In order to calculate standard errors TDA follows the advice of Birkes and Dodge [1993, p. 63], See also Dielman and Pfaffenberger [1982]. The steps are as follows.

Box 1 Syntax for `l1reg` command

```
l1reg (
     ni=1,          if without intercept, def. ni=0
     tfmt=...,      print format for results, def. 10.4
     df=...,        print data to an output file
     fmt=...,       print format for df option, def. 10.0
     ppar=...,      print estimated coefficients to output file
     pcov=...,      print unscaled covariance matrix to output file
     pres=...,      print residuals to output file
     mfmt=...,      print format for pcov and pres option, def. 12.6
     mplog=...,     write norm of residuals into matrix
     mppar=...,     write estimated parameters into matrix
     mpcov=...,     write unscaled covariance matrix
                    into matrix
) = varlist;
```

Box 2 Data file `l1reg4.dat`

```
 Y  X1  X2
----------
37   4  22
40   6  24
48   6  18
44   9  20
50  11  15
51  12   9
```

1. The residuals, $\hat{e}_i = y_i - \hat{y}_i$, are calculated and the non-zero residuals are sorted in ascending order. Further calculations are only done if the number of non-zero residuals, m, is at least 2.

2. Let k_1 and k_2 denote integers corresponding to $(m+1)/2 - \sqrt{m}$ and $(m+1)/2 + \sqrt{m}$, respectively. TDA then calculates

$$\tau = \frac{\sqrt{m}\,(\hat{e}_{(k_2)} - \hat{e}_{(k_1)})}{4}$$

   The value of τ is reported in the standard output.

3. If the data matrix, X, has full column rank, standard errors are

Box 3 Command file `l1reg4.cf`

```
nvar(
    dfile = l1reg4.dat,
    Y = c1,
    X1= c2,
    X2 = c3,
);
l1reg = Y,X1,X2;
```

Box 4 Part of standard output from `l1reg4.cf`

```
L1-norm regression.

Variables (cross-section)
-------------------------
Y   : Y
X1  : X1
X2  : X2

Solution is unique
Rank of data matrix: 3
Sum of absolute deviations: 9.7619
Number of non-zero residuals: 3

Tau:     3.7218  [calculated from residuals: -1.16667,7.42857]
Pseudorank of data matrix: 3
Norm of least squares residuals:     5.7642
Degrees of freedom: 3

Idx  Wave  Variable       Coeff      Error    Coeff/E  Signif
----------------------------------------------------------------
  1     -  Intercept    32.7143    16.8621     1.9401  0.8523
  2     1  X1            1.5952     0.9462     1.6860  0.8096
  3     1  X2           -0.0952     0.5537    -0.1720  0.1256
```

calculated as

$$\text{S.E.}(\beta_j) = \tau \sqrt{\text{diag}\{(X'X)^{-1}\}_j}$$

4. Levels of significance are calculated by using a t-distribution with $n - k$ degrees of freedom where n is the number of data points and k the number of covariates.

Example 1 To illustrate we replicate an example discussed in Birkes

and Dodge [1993, p. 66]. The command file is `l1reg4.cf`, see Box 3. Part of the standard output is shown in Box 4.

6.10.2 Nonparametric Regression

This section describes the `npreg` command for nonparametric regression with a single independent variable. The syntax is shown in Box 1. The right-hand side must provide exactly two variable names; the first one will be used for the dependent variable, the second one for the independent variable. Also required is the name of an output file, to be specified with the `df` parameter, and a list of values for the independent variable to be specified with the `x` parameter (see below). All other parameters are optional.

General Approach

Consider a 2-dimensional variable (X, Y). We seek for a regression

$$x \longrightarrow \text{Representation of } \{\Pr(Y = y \,|\, X = x)\}$$

that assigns to each value in the range of X a representation of the conditional distribution of Y given x. Our general approach will be to consider intervals

$$[x_j - d/2, x_j + d/2] \qquad \text{for} \quad j = 1, \ldots, m$$

and calculate a representation of the conditional distribution of Y for each interval separately. The interval width, $d > 0$, can be specified with the `d` parameter. The list of values, x_j, must be specified with the parameter

`x =` $x_1, \ldots, x_m$

Alternatively, one can use the syntax

`x =` $a(b)c$

The expression on the right-hand side will then be expanded into

$$x_j = a + (j-1)b \qquad \text{for} \quad j = 1, 2, \ldots$$

as long as $x_j \le c$. The output file will contain a separate record for each value x_j. The information provided will be described below, separately for each option.

Box 1 Syntax for `npreg` command

```
npreg (
     opt=...,        method selection, def. 1
                     1 : smoothed means
                     2 : smoothed quantiles
                     3 : smoothed frequencies
     k=...,          type of kernel (if opt=1), def. 1
                     1 : uniform
                     2 : triangle
                     3 : quartic
                     4 : Epanechnikov
     d=...,          band width, def. d = 1.0
     x=...,          list of points for x axis (required)
     df=...,         output file (required)
     fmt=...,        print format, def. 10.4
     dtda=...,       TDA description of output file
     sel=...,        optional case selection
) = Y,X;
```

Conditional Means

If `opt = 1`, the `npreg` command calculates conditional mean values:

$$x_j \longrightarrow \mathrm{E}(Y \mid X \in [x_j - d/2, x_j + d/2])$$

The calculation depends on the kernel selected with the `k` parameter. By default, `k=1`, the `npreg` procedure uses a uniform kernel and simply calculates the mean of all values of Y where the corresponding value of X is in the interval $[x_j - d/2, x_j + d/2]$. In this case, the command also calculates the conditional standard deviation of Y. Each record of the output file will contain the following entries:

1. The record number j (records are $j = 1, \ldots, m$).
2. The value of x_j.
3. Number of observations in $[x_j - d/2, x_j + d/2]$.
4. Mean of X in $[x_j - d/2, x_j + d/2]$.
5. Mean of Y in $[x_j - d/2, x_j + d/2]$.
6. Standard deviation of Y in $[x_j - d/2, x_j + d/2]$.

Alternative Kernels

When estimating conditional means (`opt=1`) one can use the `k` parameter to select one of the following kernel functions.[1]

1. Uniform:

$$K_d(x, x_i) := I(|\, x - x_i \,| \leq d/2)\, \frac{1}{d}$$

2. Triangle:

$$K_d(x, x_i) := I(|\, x - x_i \,| \leq d/2)\, \frac{2}{d} \left(1 - \frac{|\, x - x_i \,|}{d/2}\right)$$

3. Quartic:

$$K_d(x, x_i) := I(|\, x - x_i \,| \leq d/2)\, \frac{15}{8d} \left(1 - \left(\frac{x - x_i}{d/2}\right)^2\right)^2$$

4. Epanechnikov:

$$K_d(x, x_i) := I(|\, x - x_i \,| \leq d/2)\, \frac{3}{2d} \left(1 - \left(\frac{x - x_i}{d/2}\right)\right)^2$$

Given one of these kernels and assuming data given by (X_i, Y_i), for $i = 1, \ldots, n$, the conditional mean is calculated as

$$\hat{\mathrm{E}}(Y \mid X = x) = \frac{\sum_{i=1}^{n} K_d(x_j, X_i) Y_i}{\sum_{i=1}^{n} K_d(x_j, X_i)}$$

Note that calculation of the conditional standard deviation of the dependend variable is always based on a uniform kernel.

Conditional Quantiles

If `opt = 2`, the `npreg` command calculates conditional quantiles:

$$x_j \longrightarrow \mathrm{Q}_p(Y \mid X \in [x_j - d/2, x_j + d/2])$$

This is done for

$$p = 0.1, 0.2, 0.25, 0.3, 0.4, 0.5, 0.6, 0.7, 0.75, 0.8, 0.9$$

Each record of the output file will contain the following entries:

[1] See, e.g., Härdle [1991] for a discussion of kernel-based regression methods.

1. The record number j (records are $j = 1, \ldots, m$).
2. The value of x_j.
3. Number of observations in $[x_j - d/2, x_j + d/2]$.
4. Mean of X in $[x_j - d/2, x_j + d/2]$.
5. Then follow the conditional quantiles (11 values corresponding to the list of p-values given above).

Conditional Frequencies

If `opt = 3`, the `npreg` command sorts the values of the dependent variable in ascending order, say

$$y_1^*, \ldots, y_K^*$$

and then calculates conditional frequencies

$$x_j \longrightarrow \Pr(Y = y_k^* \mid X \in [x_j - d/2, x_j + d/2])$$

for $k = 1, \ldots, K$. Each record of the output file will contain the following entries:

1. The record number j (records are $j = 1, \ldots, m$).
2. The value of x_j.
3. Number of observations in $[x_j - d/2, x_j + d/2]$.
4. Mean of X in $[x_j - d/2, x_j + d/2]$.
5. Then follow K entries containing the proportions

$$\Pr(Y = y_k^* \mid X \in [x - d/2, x + d/2])$$

Example 1 To illustrate we replicate an example discussed by Härdle [1991, p. 124]. Box 2 shows the command file. The first `nvar` command creates the data, defined by the function

$$y_i = \sin(2\pi x_i^3)^3 + \epsilon_i \qquad \text{for} \quad i = 1, \ldots, 256$$

where the x_i are uniformely distributed in $[0, 1]$, and ϵ_i is normally distributed with $\sigma^2 \approx 0.1$. A scatterplot of the data is shown in Figure 1.

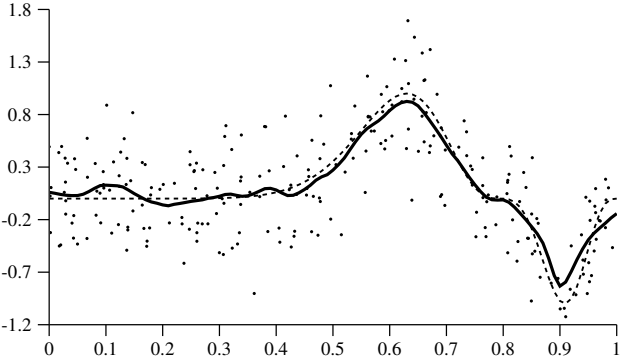


Figure 1 Plot created with `npreg1.cf` (Example 1).

The function used to create the data (omitting the noise) is shown by a dashed line.

Then follows the `npreg` command. We have selected conditional means (`opt=1`) with a quartic kernel (`k=3`), interval width is $d = 0.1$. The estimated values are written into an output file, `d.out`. Finally, the current data matrix is cleared, a new data matrix is created by reading the data file `d.out`, and the regression line is put as an additional plot object into the plot output file. Figure 1 shows the estimated regression line as a solid curve.

Box 2 Command file `npreg1.cf`

```
nvar(               # creates some random data
    noc = 256,
    X = rd,
    E = 0.32 * rdn,
    Z = sin(2 * pi * X * X * X),
    Y = Z * Z * Z + E,
);
psfile = npreg1.ps;   plot output file
psetup(
    pxlen =  90,       # length of x axis in mm
    pylen =  50,       # length of y axis in mm
    pxa   =  0,1,      # user coordinates on x axis
    pya   =  -1.2,1.8, # user coordinates on y axis
);
plxa (sc=0.1,ic=0);      plot x axis
plya (sc=0.5,ic=0);      plot y axis

plot(                   # scatterplot
    s=5,
    fs = 0.5,
    lw = 0,
) = X,Y;

# plot the function without noise

plotf (rx=0(0.01)1,lw=0.3,lt=5) = ss = sin(2 * pi * x * x * x),
                                  fn = ss * ss * ss;

npreg(              # nonparametric regression
    opt = 1,        # conditional means
    k = 3,          # quartic kernel
    d = 0.1,
    df = d.out,
    x = 0(0.01)1,
    dtda = t,
) = Y,X;

clear;              # delete data matrix
nvar(               # read data created by npreg command
  dfile = d.out,
  X     = c2 , # x value
  YM    = c5 , # mean of Y
);
plot(lw=0.5) = X,YM;    plot regression curve
```

6.11 Maximum Likelihood Estimation

Estimation of many models in TDA is based on maximum likelihood. In general, this requires finding the maximum of the model's likelihood by an iterative procedure. This chapter begins with a short introduction and then explains TDA's `fml` command that can be used to maximize user-defined log-likelihood functions. For an introduction to the mathematical and algorithmic background see **5.6** on function minimization.

6.11.1 Introduction

The starting point of ML estimation is a hypothesis involving the distribution of a random vector, say Y. The hypothesis is parametric, meaning that it depends on an unknown parameter vector, say $\theta = (\theta_1, \ldots, \theta_q)'$, but is otherwise known. This allows to derive the distribution of Y, depending on θ. For any choice of θ, the hypothesis implies a specific density of Y, say $f(y, \theta)$.

Evaluation of the hypothesis is based on a set of observations of Y. We shall assume that there is a random sample of $i = 1, \ldots, n$ independent observations y_i, realizations of corresponding sample variates, Y_i. Given the sample of observations, its likelihood and log-likelihood are defined by

$$\begin{aligned} \mathcal{L}(y_1, \ldots, y_n, \theta) &= \prod_{i=1}^{n} f(y_i, \theta) \qquad (1) \\ \ell(y_1, \ldots, y_n, \theta) &= \sum_{i=1}^{n} \log\left(f(y_i, \theta)\right) \end{aligned}$$

The ML principle simply states that a good estimator of θ can be found by maximizing this likelihood or, equivalently, the log-likelihood function. We will denote this estimator by $\hat{\theta}_n$.

The existence of the maximum likelihood estimator can be shown under very general conditions, see, e.g., White [1982]. However, the estimator is not necessarily unique. This depends on the hypothesis (model) used to derive the likelihood function and cannot be shown in general. If, for instance, the likelihood function contains two linear dependent variables, there is no longer a unique maximum. In what follows we will assume that the likelihood function has a unique global maximum.

Justification of the ML estimator is often based on asymptotic considerations. To investigate its properties, the estimator is interpreted as a function of the sample variates

$$\hat{\theta}_n = \hat{\theta}_n(Y_1, \ldots, Y_n) \qquad (2)$$

and it is assumed that there is a density of Y, say $g(y)$, according to which the observed data have been generated. The problem is to compare $g(y)$ with $f(y, \hat{\theta}_n)$, the estimated density based on maximum likelihood.

Standard maximum likelihood theory is based on the assumption that the family of distributions, given by $f(y, \theta)$, contains as a special case the unknown distribution which has generated the data. Consequently, there is a true parameter vector, say θ°, so that $g(y) = f(y, \theta^\circ)$. Given this assumption, it can be shown under very general conditions that $\hat{\theta}_n$ is a consistent estimator of θ°.

However, in almost all practical work it is very difficult, or impossible, to justify the assumption of a correct model specification. An interesting justification of the ML estimator for situations where it is unknown whether the assumption of a correct model specification holds has been given by White [1982]. With reference to Akaike [1973], he shows how ML estimation can be interpreted, then, as a method to minimize our ignorance about the distribution that has generated the data.[1] The reasoning is based on the Kullback-Leibler Information Criterion (KLIC), defined by

$$I(g, f, \theta) = \mathrm{E}\left[\log(g(Y)/f(Y, \theta))\right] \tag{3}$$

with expectation taken with respect to $g(y)$. Given that $I(g, f, \theta)$ has a unique minimum, say θ^*, it can be shown that $\hat{\theta}_n$ is a strongly consistent estimator for θ^*.[2] Furthermore, if there exists a θ° so that $g(y) = f(y, \theta^\circ)$, it follows that $\theta^* = \theta^\circ$.

This generalization of the classical ML approach has an important implication for the question how to find an appropriate estimate of the limiting distribution of $\hat{\theta}_n$. In any case, as shown by White, the difference $\hat{\theta}_n - \theta^*$ follows asymptotically a normal distribution with mean zero:

$$\sqrt{n}(\hat{\theta}_n - \theta^*) \overset{A}{\sim} N(0, \Delta) \tag{4}$$

However, there are different ways to estimate the covariance matrix, Δ. To show the alternatives, we shall use the following definitions.

$$J_n(\theta) = \left[\sum_{i=1}^{n} \frac{\partial \log(f(Y_i, \theta))}{\partial \theta_j} \frac{\partial \log(f(Y_i, \theta))}{\partial \theta_k}\right]_{j,k=1,\ldots,q} \tag{5}$$

$$H_n(\theta) = \left[\sum_{i=1}^{n} \frac{\partial^2 \log(f(Y_i, \theta))}{\partial \theta_j \, \partial \theta_k}\right]_{j,k=1,\ldots,q}$$

[1] Of course, there is no way to transcend the horizon defined by assuming that at least one member of $f(y, \theta)$ comes close to $g(y)$.

[2] Actually some additional but very mild assumptions are required, see White [1982].

$$J(\theta) = \left[\mathrm{E} \left\{ \frac{\partial \log(f(Y,\theta))}{\partial \theta_j} \frac{\partial \log(f(Y,\theta))}{\partial \theta_k} \right\} \right]_{j,k=1,\ldots,q} \tag{6}$$

$$H(\theta) = \left[\mathrm{E} \left\{ \frac{\partial^2 \log(f(Y,\theta))}{\partial \theta_j \, \partial \theta_k} \right\} \right]_{j,k=1,\ldots,q}$$

These are symmetrical (q, q) matrices. $J_n(\theta)/n$ is the sample mean, and $J(\theta)$ is the expectation (again taken with respect to $g(y)$) of the outer product of the gradient of the log-likelihood; and $H_n(\theta)/n$ is the sample mean, and $H(\theta)$ is the expectation of the Hessian of the log-likelihood.

Given these definitions, the covariance matrix of the limiting distribution (4) is:[3]

$$\Delta = H(\theta^*)^{-1} J(\theta^*) H(\theta^*)^{-1} \tag{7}$$

and, furthermore,

$$n \, H_n(\theta^*)^{-1} J_n(\theta^*) H_n(\theta^*)^{-1} \xrightarrow{a.s.} \Delta \tag{8}$$

This shows that the left-hand side of (8) which is based on sample means provides a consistent estimator of the covariance matrix, Δ.

This holds true even if the model is not correctly specified. Therefore, the left-hand side of (8) is sometimes called a robust covariance estimator. On the other hand, the classical ML approach can be shown to be a special case. One needs the assumption that there is a θ° so that $g(y) = f(y, \theta^\circ)$ and some additional regularity conditions. It follows, then, that $\theta^* = \theta^\circ$ and

$$-H(\theta^\circ) = J(\theta^\circ) \quad \text{and} \quad \Delta = -n \, H(\theta^\circ)^{-1} = n \, J(\theta^\circ)^{-1} \tag{9}$$

Summing up, there are three different ways to approximate the covariance matrix of an ML estimator. In TDA, this can be selected with the `ccov` parameter as follows:[4]

$$J_n(\hat{\theta}_n)^{-1} \quad \text{if} \quad \text{ccov} = 1 \tag{10}$$

$$-H_n(\hat{\theta}_n)^{-1} \quad \text{if} \quad \text{ccov} = 2 \tag{11}$$

$$H_n(\hat{\theta}_n)^{-1} J_n(\hat{\theta}_n) H_n(\hat{\theta}_n)^{-1} \quad \text{if} \quad \text{ccov} = 3 \tag{12}$$

[3] If the log-likelihood has a maximum it can be assumed that, at least in a neighborhood of this maximum, the matrices defined in (6) are negative definite.

[4] Note that the program calculates estimated covariance matrices for $\hat{\theta}_n - \theta^*$.

For some models, the `ccov` option is not yet implemented and TDA then uses the default, that is, `ccov=2`. In any case, the `ccov` parameter is independent of the numerical algorithm used to maximize the log-likelihood. Having found a (local) maximand, $\hat{\theta}_n$, of the log-likelihood with whatever method, TDA uses this estimate of the parameter vector in an additional evaluation of the log-likelihood and its derivatives to calculate the requested covariance estimator.

6.11.2 The `fml` Command

TDA offers the `fml` command to find parameter estimates for a user-defined log-likelihood. The syntax and all parameters are the same as in the `fmin` command described in 5.6.1. The only difference is that the default scaling factor is -1 for the `fml` command, while it is +1 for the `fmin` command. The `fml` command needs a log-likelihood function on its right-hand side. Its defintion must follow the syntax for functions explained in 5.3.1.

Example 1 To illustrate, we use our standard example data set for episode data and estimate a simple exponential model (see 6.17.2.1). The command file, `fml1.cf`, is shown in Box 1. The `nvar` command reads the episode data file `rrdat.1` and defines variables. Then comes the `edef` command to define an episode data structure. The `rate` command estimates an exponential transition rate model using `TDA`'s standard ML procedures. Then, the same model is estimated with the `fml` command. Part of the standard output is shown in Box 2. The TDA example archive contains a second example, `fml2.cf`, where the same model is estimated by using numerical integration for the cumulated transition rate.

Box 1 Command file `fml1.cf`

```
   nvar(

      dfile = rrdat.1,      # data file

      ID     [3.0] = c1,    # identification number
      SN     [2.0] = c2,    # spell number
      TS     [3.0] = c3,    # starting time
      TF     [3.0] = c4,    # ending time
      SEX    [2.0] = c5,    # sex (1 men, 2 women)
      TI     [3.0] = c6,    # interview date
      TB     [3.0] = c7,    # birth date
      TE     [3.0] = c8,    # entry into labor market
      TMAR   [3.0] = c9,    # marriage date (0 if no marriage)
      PRES   [3.0] = c10,   # prestige of current job
      PRESN  [3.0] = c11,   # prestige of next job
      EDU    [2.0] = c12,   # highest educational attainment

      # define additional variables

      COHO1 = ge(TB,348) & le(TB,384),  # birth cohort 1
      COHO2 = ge(TB,468) & le(TB,504),  # birth cohort 2
      COHO3 = ge(TB,588) & le(TB,624),  # birth cohort 3
      W     = SEX[2],                   # women = 1

      DES [1.0] = if eq(TF,TI) then 0 else 1,
      DUR [3.0] = TF - TS + 1,
   );
   edef(            # define single episode data

      ts = 0,       # starting time
      tf = DUR,     # ending time
      org = 0,      # origin state
      des = DES,    # destination state
   );

   rate(
      xa(0,1) = COHO2,COHO3,W,
   ) = 2;

   fml (
      xp = -4,0,0,0,      # starting values

   ) = rate = exp(a0 + COHO2 * a1 + COHO3 * a2 + W * a3),
      l1    = if(DES,log(rate),0),
      fn    = l1 - rate * DUR;
```

Box 2 Part of standard output from `fml1.cf`

```
ML estimation of user-defined model.

Function definition:
rate    = exp(a0+COHO2*a1+COHO3*a2+W*a3)
l1      = if(DES,log(rate),0)
fn      = l1-rate*DUR

Function will be summed over 600 data matrix cases.

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Number of model parameters: 4
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Got starting values from xp parameter.
Idx  Parameter  Starting value
  1  a0              -4.0000
  2  a1               0.0000
  3  a2               0.0000
  4  a3               0.0000

Convergence reached in 6 iterations.
Number of function evaluations: 7 (7,7)

Maximum of log likelihood: -2475.44
Norm of final gradient vector: 1.10499e-11
Last absolute change of function value: 3.30667e-15
Last relative change in parameters: 5.84648e-07

Idx  Parameter       Value      Error    Value/E  Signif
----------------------------------------------------------
  1  a0            -5.0114     0.0843   -59.4446  1.0000
  2  a1             0.5341     0.1120     4.7686  1.0000
  3  a2             0.6738     0.1152     5.8472  1.0000
  4  a3             0.5065     0.0942     5.3746  1.0000

Log likelihood (starting values): -2578.9484
Log likelihood (final estimates): -2475.4383
```

6.12 Quantal Response Models

This chapter deals with models for a dependent variable with two or more discrete categories. The models can be used with cross-sectional and pooled panel data.

6.12.1 The qreg Command

A single command, `qreg`, is used to estimate all quantal response models. The syntax is shown in Box 1. The command expects a list of variables on the right-hand side, other parameters are optional.

Cross-section data. For cross-section data, the variables on the right-hand side of the command should be specified as

$$\texttt{qreg (parameter) = } Y, X_1, X_2, \ldots;$$

First comes the name of the dependent variable, then follow the names of independent variables. If the list of independent variables is empty, the model is estimated with a single parameter for an intercept.

Panel data. The `qreg` command can also be used with panel data based on a horizontal data organization, see **3.2**. Assuming w waves, variables must be specified in the following order:

$$\begin{aligned}\texttt{qreg (parameter)} = {} & Y_1, \ldots, Y_w, \\ & X_{11}, \ldots, X_{1w}, \\ & X_{21}, \ldots, X_{2w}, \\ & \ldots, \\ & X_{m1}, \ldots, X_{mw};\end{aligned}$$

Using panel data requires to specify the number of waves with the `nw` parameter. By default, `nw=1`, that is, the command expects cross-sectional data.

As a general convention, cases are considered valid if the dependent variable has a non-negative value. This allows to use unbalanced panel data. If variable Y_t for wave t has a negative value for individual i, this wave-specific data is omitted from the likelihood. For panel data, an additional option is provided by the `pmin` parameter. If `pmin=`k, the likelihood only includes individuals who have valid data for at least k waves. Default is `pmin=1`. The number of different categories in the dependent variable is calculated from the set of non-zero values found in at least one of these variables. These values are always taken as integers and then sorted in ascending order. The smallest integer is taken as the reference

Box 1 Syntax for `qreg` command

```
qreg(
     m=...,          model selection, def. 1
                     1 = binary logit
                     2 = binary probit
                     3 = ordinal logit
                     4 = ordinal probit
                     5 = multinomial logit
                     6 = multivariate probit
                     7 = conditional logit
                     8 = simultaneous binary probits
     ni=1,           if without intercept, def. 0
     nq=...,         number of categories if m=5
     nw=...,         number of waves, def. 1
     pmin=...,       minimum number of valid wave data, def. 1
     tfmt=...,       print format for results, def. 10.4
     ppar=...,       print estimated coefficients to output file
     pcov=...,       print covariance matrix to output file
     res=...,        additional output options
     mfmt=...,       print format for pcov option, def. 12.4
     df=...,         print data to an output file
     dtda=...,       TDA description of output file
     fmt=...,        print format for df option, def. 10.4
     opt=...,        model-specific options, def. 1
     nhp=...,        control of numerical integration, def. 6
     eps=...,        accuracy of numerical integration, def. 10^-6
     mplog=...,      write loglikelihood value into matrix
     mppar=...,      write estimated parameters into matrix
     mpcov=...,      write covariance matrix into matrix
     mpgrad=...,     write gradients into matrix
) = varlist;
```

category. The maximum number of categories can be specified with the `maxcat` parameter; default is `maxcat=100`.

Additional options. Model estimation is done with the maximum likelihood method. Most parameters discussed in 5.6.1 for the `fmin` command can also be used for the `qreg` command. This includes the selection of a minimization algorithm and parameter constraints.

1. Results of model estimation are written into the standard output. The print format can be controlled with the `tfmt` parameter, default is `tfmt=10.4`. For a description of the output see the examples given in subsequent sections.

2. The parameter

```
ppar = name_of_an_output_file,
```

can be used to write the estimated model coefficients (and standard errors) into an output file. The print format is determined by `tfmt`.

3. By default, the covariance matrix is calculated from the matrix of second derivatives of the log-likelihood function. One can use the `ccov` parameter to require an alternative calculation procedure; see the discussion in **5.6.5** and **6.11.1**.

4. The parameter

```
pcov = name_of_an_output_file,
```

can be used to write the estimated covariance matrix into an output file. The print format can be controlled with the `mfmt` parameter, default is `mfmt=12.4`.

5. The data used for model estimation can be written into an output file with the `df` parameter. For most models, the output file will then also include estimated probabilities for each individual case. Using the `dtda` parameter provides a description of the output file corresponding to the syntax of TDA. The print format can be controlled with the `fmt` parameter.

6. Some of the additional parameters are model-specific and will be explained in the corresponding sections.

6.12.2 Binary Logit and Probit

This section describes simple logit and probit models for cross-sectional or pooled panel data. It is assumed that the dependent variable, Y, has only two possible values, $Y \in \{0, 1\}$. The 0-1 coding is actually only used to ease notation, any other non-negative integer values may be used. TDA always sorts the categories of the dependent variable in ascending order, the sorted categories are then mapped to 0 and 1. Choosing category 0 to be the reference category, the modeling approach is

$$\Pr(Y = 1 \mid x) = F(x; \beta) \tag{1}$$

F is a suitably chosen function depending on a parameter vector β and a (row) vector of covariates, x. For each possible realization of these covariates, the model implies an estimate of the probability that the dependent variable takes the value 1.

The function F can be specified in many different ways; each specification results in a different type of model. Most commonly, the specification is in terms of a cumulative distribution function; this guarantees its values to be in the range of zero to one, required for an interpretation as probabilities. If the distribution function has only a single parameter, the dependence on covariates is modeled by a single predictor, most commonly assumed to be linear, i.e., $x\beta$. Of course, in principle it would also be possible to use more general distribution functions with two or more parameters and then linking covariates differently to all of these parameters.

Two choices of F are most popular. One is the distribution function of the logistic distribution

$$F(y) = \frac{\exp(y)}{1 + \exp(y)}$$

Choosing this distribution function leads to a logit model:

$$\Pr(Y = 1 \mid x) = \frac{\exp(x\beta)}{1 + \exp(x\beta)} \tag{2}$$

Another popular choice is the standard normal distribution function

$$\Phi(x) = \int_{-\infty}^{x} \phi(u)\, du \qquad \text{with} \quad \phi(u) = \frac{1}{\sqrt{2\,\pi}} \exp(-\frac{1}{2}\, u^2) \tag{3}$$

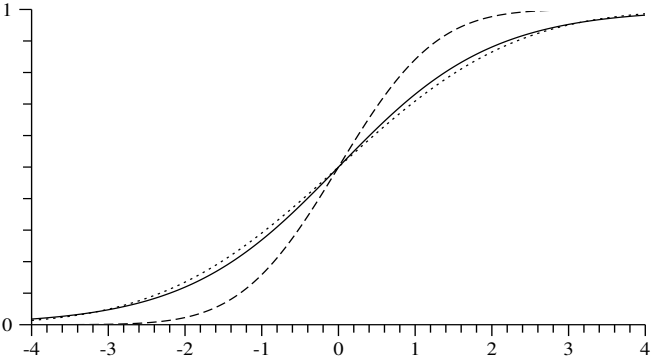


Figure 1 Normal and logistic distribution functions.

Choosing the normal distribution function to specify (1) leads to a probit model:

$$\Pr(Y = 1 \mid x) = \Phi(x\beta) \tag{4}$$

Both distributions are very similar as can be seen in Figure 1.[1] The solid and dashed lines show the standard logistic and the standard normal distribution functions, respectively. The difference is mainly due to a different variance. The standard logistic distribution has variance $\pi^2/3$. A normal distribution function with this variance is plotted as a dotted line in Figure 1.

However, in both models it is not possible to include explicitly a separate parameter for the variance. Taking $\Phi(x\beta/\sigma)$ in (4), or $\exp(\sigma x\beta)/(1+\exp(\sigma x\beta))$ in (2), shows that σ is not identified. In both, the logit and probit model, the parameter vector β is identified only up to a multiplicative scalar.

Maximum Likelihood Estimation. Using y_i to denote observed values of the dependent variables, the likelihood of a sample of observations can be written

$$\mathcal{L} = \prod_{i=1}^{N} \Pr(Y_i = 1 \mid x_i)^{y_i} \Pr(Y_i = 0 \mid x_i)^{1-y_i} \tag{5}$$

Case weights will be used if defined with the `cwt` command.

[1] The plot was created with command file `plot-bin.cf` contained in the TDA example archive.

Logit Model. Inserting the logit specification (2) and taking logarithms gives the log-likelihood of the logit model:

$$\begin{aligned}\ell &= \sum_{i=1}^{N} y_i \log\left\{\frac{\exp(x_i\beta)}{1+\exp(x_i\beta)}\right\} + (1-y_i)\log\left\{\frac{1}{1+\exp(x_i\beta)}\right\} \\ &= \sum_{i=1}^{N} y_i\, x_i\beta - \log\{1+\exp(x_i\beta)\}\end{aligned}$$

This log-likelihood is well behaved (globally concave) and can be maximized easily with the standard Newton algorithm. The first and second derivatives with respect to the components of the parameter vector, β, are

$$\begin{aligned}\frac{\partial \ell}{\partial \beta_j} &= \sum_{i=1}^{N}\left\{y_i - \frac{\exp(x_i\beta)}{1+\exp(x_i\beta)}\right\} x_{ij} \\ \frac{\partial^2 \ell}{\partial \beta_j\, \partial \beta_k} &= \sum_{i=1}^{N} \frac{-\exp(x_i\beta)}{(1+\exp(x_i\beta))^2}\, x_{ij}\, x_{ik}\end{aligned}$$

Here it is assumed that there are $j = 1, \ldots, m$ covariates, x_{ij} being the jth covariate for individual i. If the model is specified to include an intercept (default), x_{i1} is a constant one.

Probit Model. Inserting the probit specification (4) into (5) and taking logarithms gives the log-likelihood for the probit model:

$$\ell = \sum_{i=1}^{N} y_i \log\{\Phi(x_i\beta)\} + (1-y_i)\log\{1-\Phi(x_i\beta)\}$$

The first and second derivatives with respect to the model parameters are

$$\begin{aligned}\frac{\partial \ell}{\partial \beta_j} &= \sum_{i=1}^{N}\left\{y_i \frac{\phi(x_i\beta)}{\Phi(x_i\beta)} - (1-y_i)\frac{\phi(x_i\beta)}{1-\Phi(x_i\beta)}\right\} x_{ij} \\ \frac{\partial^2 \ell}{\partial \beta_j\, \partial \beta_k} &= \sum_{i=1}^{N}\left\{y_i \frac{\phi(x_i\beta)}{\Phi(x_i\beta)}\left[-x_i\beta - \frac{\phi(x_i\beta)}{\Phi(x_i\beta)}\right] + \right. \\ &\qquad \left. (1-y_i)\frac{\phi(x_i\beta)}{1-\Phi(x_i\beta)}\left[x_i\beta - \frac{\phi(x_i\beta)}{1-\Phi(x_i\beta)}\right]\right\} x_{ij}\, x_{ik}\end{aligned}$$

Box 1 Data file `qr1.dat`

```
D    W  R
---------
1    9  0
1    1  1
2   10  0
2    2  1
3    6  0
3    4  1
4    5  0
4    5  1
5    4  0
5    8  1
6    2  0
6    8  1
7   10  1
```

As in (3), ϕ denotes the standard normal density function; x_{ij} denotes the observed value of the jth covariate for the ith individual.

Implementation. Both, the binary logit and probit model can be estimated with the `qreg` command; the `m=1` parameter selects the logit model, the `m=2` parameter selects the probit model.

Model estimation is done by maximum likelihood as described above. The default algorithm is `mina=5` (Newton I). It is possible to use the `ccov` option to select a specific type of covariance matrix calculation; default is `ccov=2`, that is, the covariance matrix is create from the Hessian of the log-likelihood. The parameter vector, and accordingly the first and second derivatives of the log-likelihood, is organized as

$$\boxed{\beta_1 \ \ldots \ \beta_m}$$

with m the number of covariates. As default, the first parameter is added automatically as an intercept. The `ni=1` option can be used to estimate a model without an intercept. Since the log-likelihood for the simple logit and probit models is globally concave, starting values for its iterative maximization are, in most cases, not critical. As a default, TDA sets all starting values to zero.

Example 1 To illustrate estimation of logit and probit models, we use a small example data set taken from the SAS User's Guide (Version 6, vol. II, p. 1340). The data file, `qr1.dat`, is shown in Box 1. Each row of the

Box 2 command file `qr1.cf`

```
nvar(
    dfile = qr1.dat,

    Dose = c1,
    Weight = c2,
    Response = c3,
    Log10Dose = log(Dose) / log(10),
);
cwt = Weight;   use case weights

qreg = Response,Log10Dose;          logit model
qreg (m=2) = Response,Log10Dose;    probit model

# fml command to estimate same logit model

fml = xb = beta0 + Log10Dose * beta1,
      fn = Weight * (Response * xb - log(1 + exp(xb)));

# fml command to estimate same probit model

fml = pxb = nd (beta0 + Log10Dose * beta1),
      fn  = Weight * (Response * log(pxb) +
                      (1 - Response) * log(1 - pxb));
```

data file corresponds to a group of individuals with the stimulus variable recorded in `Log10Dose`, the response variable recorded in `Response`, and the number of individuals given in `Weight`.

The command file `qr1.cf`, shown in Box 2, estimates a logit and a probit model. First this is done with the `qreg` command, then with TDA's `fml` command that requires a user-defined log-likelihood (see **6.11.2**). Part of the standard output of the `qreg` commands is shown in Box 3 (logit model) and Box 4 (probit model).

Standardized Effect Coefficients. As described by Long [1987], to ease interpretation of estimation results of binary (and multinomial) logit models, it is helpful to calculate standardized effect coefficients, defined by

$$\text{standardized effect of } j\text{th covariate} \;=\; \exp(\hat{\beta}_j\, \hat{\sigma}(X_j))$$

$\hat{\sigma}(X_j)$ is the standard deviation of covariate X_j. A calculation of these standardized effects can be requested with the parameter

```
res = 1,
```

Box 3 Part of standard output from `qr1.cf` (logit model)

```
Model: binary logit.

Variables (cross-section)
-------------------------
Y   : Response
X1  : Log10Dose

Checking available data (pmin=1)
Number of cases with valid data: 13

Categories of dependent variable.
Maximum number of categories: 100

Index               0         1   (Weighted)
Category            0         1  Observations
-----------------------------------------------
Wave 1   N      36.00     38.00         74.00
         Pct    48.65     51.35

Using case weights defined by: Weight
Maximum likelihood estimation.
Algorithm 5: Newton (I)

Number of model parameters: 2
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Convergence reached in 6 iterations.
Number of function evaluations: 7 (7,7)

Maximum of log likelihood: -37.1107
Norm of final gradient vector: 1.5979e-10
Last absolute change of function value: 9.82354e-12
Last relative change in parameters: 7.41904e-06

Idx Cat Term   Variable       Coeff      Error    C/Error  Signif
-----------------------------------------------------------------
  1   1  I     Intercept    -3.2246     0.8861    -3.6393  0.9997
  2   1  X     Log10Dose     5.9702     1.4492     4.1197  1.0000

Log likelihood (starting values):   -51.2929
Log likelihood (final estimates):   -37.1107
```

Box 4 Part of standard output from `qr1.cf` (probit model)

```
Model: binary probit.

Convergence reached in 5 iterations.
Number of function evaluations: 6 (6,6)

Maximum of log likelihood: -37.2804
Norm of final gradient vector: 3.44131e-07
Last absolute change of function value: 2.0696e-08
Last relative change in parameters: 0.00030733

Idx Cat Term   Variable       Coeff      Error    C/Error  Signif
-------------------------------------------------------------------
  1   1  I     Intercept    -1.8127     0.4493    -4.0341  0.9999
  2   1  X     Log10Dose     3.4181     0.7455     4.5847  1.0000

Log likelihood (starting values):   -51.2929
Log likelihood (final estimates):   -37.2804
```

Box 5 Standardized coefficients (logit model)

```
Standardized coefficients.

Idx Cat Term   Variable       Coeff     Exp(C)  Exp(C*SD)   Std.Dev.
-------------------------------------------------------------------
  1   1  I     Intercept    -3.2246     0.0398     1.0000     0.0000
  2   1  X     Log10Dose     5.9702   391.5761     1.0000     0.0000
```

To illustrate, adding this command to command file `qr1.cf` creates the table shown in Box 5. Standardized effects are only calculated for cross-sectional input data (`nw=1`).

Estimated Probabilities. The `df` parameter can be used to request an output file that contains the data used for model estimation and, in addition, the estimated probabilities. An illustration is given in Box 6. The columns labelled `PROB0` and `PROB1` contain the estimated probabilities for Y to take the values 0 and 1, respectively, conditional on the corresponding values of the covariates, calculated as $F(x\beta)$ and $1 - F(x\beta)$, respectively.

The column labelled `PROB` is the probability for the actually observed value of the dependent variable. The data in Box 6 have been created

Box 6 Illustration of `df` and `dtda` parameters

```
Contents of file requested by dtda parameter

nvar(
  dfile = df,
  noc = 13,
  CaseID    [6.0] = c1 ,
  Wave      [2.0] = c2 ,
  Response  [10.4] = c3 ,
  Log10Dose [10.4] = c4 ,
  Weight    [10.4] = c5 ,
  PROB      [10.4] = c6 ,
  PROB0     [10.4] = c7 ,
  PROB1     [10.4] = c8 ,
);

Contents of file requested by df parameter

 1  1   0.0000   0.0000   9.0000   0.9618   0.9618   0.0382
 2  1   1.0000   0.0000   1.0000   0.0382   0.9618   0.0382
 3  1   0.0000   0.3010  10.0000   0.8065   0.8065   0.1935
 4  1   1.0000   0.3010   2.0000   0.1935   0.8065   0.1935
 5  1   0.0000   0.4771   6.0000   0.5929   0.5929   0.4071
 6  1   1.0000   0.4771   4.0000   0.4071   0.5929   0.4071
 7  1   0.0000   0.6021   5.0000   0.4086   0.4086   0.5914
 8  1   1.0000   0.6021   5.0000   0.5914   0.4086   0.5914
 9  1   0.0000   0.6990   4.0000   0.2792   0.2792   0.7208
10  1   1.0000   0.6990   8.0000   0.7208   0.2792   0.7208
11  1   0.0000   0.7782   2.0000   0.1945   0.1945   0.8055
12  1   1.0000   0.7782   8.0000   0.8055   0.1945   0.8055
13  1   1.0000   0.8451  10.0000   0.8607   0.1393   0.8607
```

by adding the parameters

```
df   = name_of_an_output_file,
dtda = name_of_an_output_file,
```

to the `qreg` command for a logit model (see command file `qr1a.cf`, not shown).

6.12.2.1 Grouped Data

Assume that the sample consists of m groups and the ith group has n_i members. Let y_i denote the number of members in group i who have $Y = 1$. The likelihood may then be written as

$$\mathcal{L} = \prod_{i=1}^{m} \binom{n_i}{y_i} p_i^{y_i} (1 - p_i)^{n_i - y_i} \tag{1}$$

where p_i is the probability for $Y = 1$ in group i, see Collett [1991, p. 57]. If one not just observes n_i and y_i, but also group-specific vectors of covariates, say x_i, one might go on and model the dependence of p_i on x_i. For example, using a logit specification, one can assume

$$p_i = \frac{\exp(x_i\beta)}{1 + \exp(x_i\beta)}$$

The model is easily estimated with the `qreg` command using either individual or grouped data. For example, with grouped data, one can use a data structure like

Y	W	X
1	y_i	x_i
0	$n_i - y_i$	x_i
$\vdots$	$\vdots$	$\vdots$

to be repeated for each group. The model is estimated by defining case weights with `cwt = W` and then using the command

```
qreg (parameter) = Y,X;
```

Given that the data are clustered into groups it may also make sense to estimate a multilevel model, see, e.g., Collett [1991, p. 207]. One approach is by including a random effect into the model specification. Using again a logit model, one might write

$$p_i = \frac{\exp(x_i\beta + z_i\gamma)}{1 + \exp(x_i\beta + z_i\gamma)}$$

Box 1 Data file `qr5.dat`

```
 Y  N  S  X    Y  N  S  X    Y  N  S  X    Y  N  S  X
-----------   -----------   -----------   -----------
10 39  0  0    5  6  0  1    8 16  1  0    3 12  1  1
23 62  0  0   53 74  0  1   10 30  1  0   22 41  1  1
23 81  0  0   55 72  0  1    8 28  1  0   15 30  1  1
26 51  0  0   32 51  0  1   23 45  1  0   32 51  1  1
17 39  0  0   46 79  0  1    0  4  1  0    3  7  1  1
              10 13  0  1
```

where z_i is a random number drawn from some distribution and γ an additional model parameter. In order to make the model estimable, one needs an assumption about the distribution of the random effects. Assuming a density, say $\phi(z)$, one can then try a marginal likelihood approach:

$$\mathcal{L} = \prod_{i=1}^{m} \int_{-\infty}^{\infty} \binom{n_i}{y_i} p_i^{y_i} (1-p_i)^{n_i - y_i} \phi(z_i)\, \mathrm{d@}, z_i \tag{2}$$

For a discussion of this model where it is assumed that ϕ is a standard normal density, see Collett [1991, pp. 207–212].

Example 1 While it is not possible to use the `qreg` command for multilevel models one can try to set up the likelihood directly and then use the `fml` command, see **6.11.2**. To illustrate this approach we use an example from Collett [1991, p. 211]. The data are shown in Box 1. There are 21 groups. `N` is the number of group members, and `Y` is the number of individuals with $Y = 1$. `S` and `X` are covariates (four categories).

The command file is `qr5.cf`, shown in Box 2. The first `fml` command uses the likelihood (1). Since the likelihood is well-behaved, there is no need to specify starting values (as default, all are zero), and one can use TDA's default Newton algorithm for function minimization. First and second derivatives are automatically calculated as discussed in **5.3.2**. Box 3 shows part of the standard output. As can be seen from this output, convergence is reached in four iterations.

To estimate a random effects model is far more difficult since we then need numerical integration and do not know much about the behavior of the likelihood. The second `fml` command in command file `qr5.cf` uses a direct translation of the likelihood shown in (2). Creating the marginal likelihood is done with the `intn` operator that uses Hermite integration

Box 2 Command file `qr5.cf`

```
nvar(
    dfile = qr5.dat,
    Y = c1,
    N = c2,
    S = c3,
    X = c4,
    SX = S * X,
);

# standard logit model

fml = xb = beta0 + S * beta1 + X * beta2 + SX * beta3,
      ee = bc(N,Y) * (exp(xb)^Y) / ((1 + exp(xb))^N) ,
      fn = log(ee);

# logit model with random effects

fml(
  mina = 4,
  xp =  -0.5582,0.1459,1.3182,-0.7781,0.1,  # starting values
)  = xb = beta0 + S * beta1 + X * beta2 + SX * beta3,
ee = intn(7,bc(N,Y) * (exp(xb+t*gam)^Y)/((1 + exp(xb+t*gam))^N) ),
fn = log(ee);
```

to calculate

$$\text{intn}(p, \text{expression}(t)) \approx \int_{-\infty}^{\infty} \text{expression}(t)\, \phi(t)\, \mathrm{d@}, t$$

where ϕ is a standard normal density and p is the number of Hermite integration points. For more information on this operator, see **5.4.2**.

Since the likelihood is difficult to maximize, we use starting values taken from the estimation results of the simple logit model. Also, we use the BFGS algorithm (`mina=4`) that only requires first derivatives.[1] Nevertheless, having reached convergence, standard errors are calculated from the Hessian of second derivatives. Box 4 shows the estimation results. They are basically identical with the results found by Collett [1991, p. 212] who used the EGRET program.

[1] The performance of the minimization algorithm depends, in particular, on the number of integration points. It might be necessary to begin with a low number of integration points and then using the estimated parameters as starting values for higher number of integration points.

Box 3 Part of standard output from `qr5.cf`

```
ML estimation of user-defined model.

Function definition:
xb      = beta0+S*beta1+X*beta2+SX*beta3
ee      = bc(N,Y)*(exp(xb)^Y)/((1+exp(xb))^N)
fn      = log(ee)

Function evaluation: sum over 21 data matrix cases.

Maximum likelihood estimation.
Algorithm 5: Newton (I)

Number of model parameters: 4
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Idx  Parameter   Starting value
  1  beta0        0.00000000e+00
  2  beta1        0.00000000e+00
  3  beta2        0.00000000e+00
  4  beta3        0.00000000e+00

Convergence reached in 4 iterations.
Number of function evaluations: 5 (5,5)

Maximum of log likelihood: -54.937
Norm of final gradient vector: 7.20504e-07
Last absolute change of function value: 2.61263e-08
Last relative change in parameters: 0.000299634

Idx  Parameter       Value      Error    Value/E  Signif
----------------------------------------------------------
  1  beta0         -0.5582     0.1260    -4.4292  1.0000
  2  beta1          0.1459     0.2232     0.6539  0.4868
  3  beta2          1.3182     0.1775     7.4277  1.0000
  4  beta3         -0.7781     0.3064    -2.5392  0.9889

Log likelihood (starting values):   -87.8318
Log likelihood (final estimates):   -54.9370
```

Box 4 Part of standard output from `qr5.cf`

```
ML estimation of user-defined model.

Function definition:
xb      = beta0+S*beta1+X*beta2+SX*beta3
ee      = intn(7,bc(N,Y)*(exp(xb+t*gam)^Y)/((1+exp(xb+t*gam))^N))
fn      = log(ee)

Function evaluation: sum over 21 data matrix cases.

Maximum likelihood estimation.
Algorithm 4: BFGS

Number of model parameters: 5
Type of covariance matrix: 2
Maximum number of iterations: 100
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Minimum of step size value: 1e-10
Scaling factor: -1

Got 5 starting value(s) from xp parameter.

Idx  Parameter   Starting value
  1  beta0       -5.58200000e-01
  2  beta1        1.45900000e-01
  3  beta2        1.31820000e+00
  4  beta3       -7.78100000e-01
  5  gam          1.00000000e-01

Convergence reached in 16 iterations.
Number of function evaluations: 19 (19,1)

Maximum of log likelihood: -53.7562
Norm of final gradient vector: 6.18713e-07

Idx  Parameter       Value      Error    Value/E  Signif
-------------------------------------------------------
  1  beta0         -0.5481     0.1668    -3.2861  0.9990
  2  beta1          0.0965     0.2785     0.3464  0.2710
  3  beta2          1.3369     0.2369     5.6442  1.0000
  4  beta3         -0.8104     0.3853    -2.1034  0.9646
  5  gam            0.2368     0.1110     2.1335  0.9671

Log likelihood (starting values):   -54.4375
Log likelihood (final estimates):   -53.7562
```

6.12.3 Ordinal Logit and Probit

This section describes logit and probit models for a dependent variable with ordered categories. (For an applied introduction with social science examples see Ludwig-Mayerhofer [1990].) It is assumed that there are at least three such categories:

$$R_1 < R_2 < \ldots < R_q \qquad q \geq 3$$

The dependent variable is

$$Y \in \{R_1, R_2, \ldots, R_q\} \qquad q \geq 3 \tag{1}$$

with observed values denoted by y_i; x_i is a corresponding row vector containing, say, m covariates. There are two mathematically equivalent methods to extend the simple binary logit and probit models discussed in 6.12.2 to the case of an ordinal response variable. One approach is to formulate the model in terms of cumulative probabilities (logits or probits), the other approach uses the concept of a latent variable.

Cumulative Logits and Probits. The definition of cumulative probabilities (logits or probits) is given by

$$\theta_j = \Pr(Y > R_j) \qquad j = 1, \ldots, q$$

Of course, $\theta_q = 0$. Defining $\theta_0 = 1$, it follows that

$$\Pr(Y = R_j) = \theta_{j-1} - \theta_j \qquad j = 1, \ldots, q$$

Using these quantities, θ_j, a general probability model for an ordinal dependent variable can be written

$$\theta_j = F(\alpha_j + x\,\beta) \qquad j = 1, \ldots, q-1 \tag{2}$$

where F is a suitably chosen distribution function. The basic idea is to assume a different intercept, α_j, for each of the ordered categories of the dependent variable. In this way, the model links the covariates to the probability of reaching a given level of the dependent variable. Since $\theta_q = 0$, only $q-1$ intercepts can be estimated, and there cannot be an additional intercept in the vector of covariates, x.

Choosing F as a logistic distribution function leads to the logit model for an ordinal dependent variable:

$$\theta_j = \frac{\exp(\alpha_j + x\,\beta)}{1 + \exp(\alpha_j + x\,\beta)} \qquad j = 1, \ldots, q-1$$

The corresponding probit model is

$$\theta_j = \Phi(\alpha_j + x\,\beta) \qquad j = 1, \ldots, q-1$$

with Φ denoting the standard normal distribution function.

Formulation with a Latent Variable. Another way to derive probability models for an ordered dependent variable, discussed by McKelvey and Zavoina [1975],[1] is in terms of a continuous latent variable, say Y^*. It is assumed that there are unknown threshold values

$$-\infty = \mu_0 < \mu_1 < \ldots < \mu_{q-1} < \mu_q = \infty$$

so that there is a correspondence

$$Y = R_j \quad \Leftrightarrow \quad \mu_{j-1} < Y^* \leq \mu_j \qquad j = 1, \ldots, q$$

This approach leads to an ordinary regression model for Y^* that can be written

$$Y^* = x\,\beta + \epsilon \tag{3}$$

However, the model can equally well be written as a probability model, and then it is seen to be equivalent to the formulation given in (2).

$$\begin{aligned} \theta_j &= \Pr(Y > R_j) = \Pr(Y^* > \mu_j) \\ &= \Pr(-\epsilon < x\,\beta - \mu_j) = F(x\,\beta - \mu_j) \end{aligned} \tag{4}$$

with F the distribution function of the disturbance term, ϵ, in (3).[2] It follows that

$$\alpha_j = -\mu_j \qquad j = 1, \ldots, q-1$$

[1] The same approach has been used earlier to find a scaling procedure for ordinal categorical data, see Snell [1964].

[2] It is assumed that ϵ has zero mean and that F is symmetric.

This assumes that the covariate vector x does not contain an intercept. Of course, other parametrizations are possible.[3]

Maximum Likelihood Estimation. The likelihood of a sample of observations can be written as

$$\mathcal{L} = \prod_{i=1}^{N} \prod_{j=1}^{q} \Pr\left(Y_i = R_j \mid x_i\right)^{w_{ij}}$$

with indicator variables w_{ij} defined by

$$w_{ij} = \begin{cases} 1 & \text{if } Y_i = R_j \\ 0 & \text{otherwise} \end{cases}$$

Using cumulative logits or probits, the log-likelihood is

$$\ell = \sum_{i=1}^{N} \sum_{j=1}^{q} w_{ij} \log(\theta_{i,j-1} - \theta_{ij})$$

To maximize this expression and to estimate the covariance matrix, one needs first and second derivatives with respect to α_u $(u = 1, \ldots, q-1)$ and with respect to the components of the parameter vector, β. They are given below for the logit and probit case.

Ordinal Logit. To simplify notation, the following abbreviations are used:

$$P_{ij} = \theta_{i,j-1} - \theta_{ij}\ , \quad C_{i0} = C_{iq} = 0$$

$$C_{ij} = \frac{\theta_{ij}}{1 + \exp\left(\alpha_j + x_i\beta\right)}$$

$$D_{ij} = \frac{C_{ij}}{1 + \exp\left(\alpha_j + x_i\beta\right)} \qquad j = 1, \ldots, q-1$$

[3]LIMDEP, for instance, sets $\mu_0 = 0$ and requires an intercept in x. The correspondence is then given by $\alpha_1 =$ estimate of the global intercept, and $\alpha_j = \alpha_1 - \mu_{j-1}$.

With these abbreviations and using the Kronecker symbol, δ_{ij}, the first and second derivatives are

$$\frac{\partial \ell}{\partial \alpha_u} = \sum_{i=1}^{m}\sum_{j=1}^{q} w_{ij}\frac{\delta_{u,j-1}C_{i,j-1} - \delta_{uj}C_{ij}}{P_{ij}}$$

$$\frac{\partial \ell}{\partial \beta_u} = \sum_{i=1}^{m}\sum_{j=1}^{q} w_{ij}\frac{C_{i,j-1} - C_{ij}}{P_{ij}}x_{iu}$$

$$\begin{aligned}\frac{\partial^2 \ell}{\partial \alpha_u \partial \alpha_v} = \sum_{i=1}^{m}\sum_{j=1}^{q} w_{ij}\Bigg\{&\frac{\delta_{u,j-1}\delta_{v,j-1}(D_{i,j-1} - C_{i,j-1}\theta_{i,j-1})}{P_{ij}} - \\ &\frac{\delta_{u,j}\delta_{vj}(D_{ij} - C_{ij}\theta_{ij})}{P_{ij}} - \\ &\frac{(\delta_{u,j-1}C_{i,j-1} - \delta_{uj}C_{ij})(\delta_{v,j-1}C_{i,j-1} - \delta_{vj}C_{ij})}{P_{ij}P_{ij}}\Bigg\}\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 \ell}{\partial \beta_u \partial \beta_v} = \sum_{i=1}^{m}\sum_{j=1}^{q} w_{ij}x_{iu}x_{iv}\Bigg\{&\frac{(D_{i,j-1} - C_{i,j-1}\theta_{i,j-1})}{P_{ij}} - \\ &\frac{(D_{ij} - C_{ij}\theta_{ij})}{P_{ij}} - \left[\frac{C_{i,j-1} - C_{ij}}{P_{ij}}\right]^2\Bigg\}\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 \ell}{\partial \beta_u \partial \alpha_v} = \sum_{i=1}^{m}\sum_{j=1}^{q} w_{ij}x_{iu}\Bigg\{&\frac{\delta_{v,j-1}(D_{i,j-1} - C_{i,j-1}\theta_{i,j-1})}{P_{ij}} - \\ &\frac{\delta_{vj}(D_{ij} - C_{ij}\theta_{ij})}{P_{ij}} - \frac{(C_{i,j-1} - C_{ij})(\delta_{v,j-1}C_{i,j-1} - \delta_{vj}C_{ij})}{P_{ij}P_{ij}}\Bigg\}\end{aligned}$$

Ordinal Probit. Using the abbreviations

$$E_{ij} = \alpha_j + x_i\beta, \qquad \text{and} \qquad \Phi_{ij} = \Phi(\alpha_i + x_i\beta)$$

the first and second derivatives can be written as

$$\begin{aligned}
\frac{\partial \ell}{\partial \alpha_u} &= \sum_{i=1}^{m}\sum_{j=1}^{q} w_{ij}\,\frac{\delta_{u,j-1}\Phi_{i,j-1} - \delta_{uj}\Phi_{ij}}{P_{ij}} \\
\frac{\partial \ell}{\partial \beta_u} &= \sum_{i=1}^{m}\sum_{j=1}^{q} w_{ij}\,\frac{\Phi_{i,j-1} - \Phi_{ij}}{P_{ij}}\,x_{iu} \\
\frac{\partial^2 \ell}{\partial \alpha_u \partial \alpha_v} &= \sum_{i=1}^{m}\sum_{j=1}^{q} w_{ij}\left\{\frac{\delta_{uj}\delta_{vj}E_{ij}\Phi_{i,j} - \delta_{u,j-1}\delta_{v,j-1}E_{i,j-1}\Phi_{i,j-1}}{P_{ij}} - \right. \\
&\qquad \left. \frac{(\delta_{u,j-1}\Phi_{i,j-1} - \delta_{uj}\Phi_{ij})(\delta_{v,j-1}\Phi_{i,j-1} - \delta_{vj}\Phi_{ij})}{P_{ij}P_{ij}}\right\} \\
\frac{\partial^2 \ell}{\partial \beta_u \partial \beta_v} &= \sum_{i=1}^{m}\sum_{j=1}^{q} w_{ij}x_{iu}x_{iv}\left\{\frac{E_{ij}\Phi_{ij} - E_{i,j-1}\Phi_{i,j-1}}{P_{ij}} - \right. \\
&\qquad \left. \left[\frac{\Phi_{i,j-1} - \Phi_{ij}}{P_{ij}}\right]^2\right\} \\
\frac{\partial^2 \ell}{\partial \beta_u \partial \alpha_v} &= \sum_{i=1}^{m}\sum_{j=1}^{q} w_{ij}x_{iu}\left\{\frac{\delta_{vj}E_{ij}\Phi_{ij} - \delta_{v,j-1}E_{i,j-1}\Phi_{i,j-1}}{P_{ij}} - \right. \\
&\qquad \left. \frac{(\Phi_{i,j-1} - \Phi_{ij})(\delta_{v,j-1}\Phi_{i,j-1} - \delta_{vj}\Phi_{ij})}{P_{ij}P_{ij}}\right\}
\end{aligned}$$

Implementation. Both models can be estimated with the `qreg` command. `m=3` selects the ordinal logit, `m=4` selects the ordinal probit model. Default is cross-sectional data. If there are two or more panel waves, the data are pooled over waves. The models are parameterized using cumulative logits and probits. Estimation is done by maximum likelihood as described above.

Both models have $m + q - 1$ parameters where m is the number of covariates and q is the number of categories of the dependent variable. It

Box 1 Data file `qr2.dat`

Y	X1	X2	Y	X1	X2
2	1.163	1.000	1	1.058	1.000
2	3.453	0.000	0	2.607	1.000
2	3.724	0.000	0	2.162	0.000
0	1.090	0.000	1	2.145	0.000
3	3.819	0.000	0	1.103	0.000
1	2.838	1.000	1	1.914	0.000
3	1.516	1.000	3	3.610	0.000
3	1.417	1.000	3	3.579	1.000
0	2.436	1.000	3	3.379	1.000
2	3.300	1.000	1	3.175	0.000

Box 2 Command file `qr2.cf`

```
nvar(
    dfile = qr2.dat,
    Y  = c1,
    X1 = c2,
    X2 = c3,
);
qreg (m=3) = Y,X1,X2;     ordinal logit
qreg (m=4) = Y,X1,X2;     ordinal probit

fml(                      # ordinal logit with fml command
    xp = -2,-3,-4,0,0,

)   = xb     = X1 * beta1 + X2 * beta2,
      theta0 = 1,
      theta1 = eexp(alpha1 + xb),
      theta2 = eexp(alpha2 + xb),
      theta3 = eexp(alpha3 + xb),
      ll     = if (eq(Y,0),theta0 - theta1,
               if (eq(Y,1),theta1 - theta2,
               if (eq(Y,2),theta2 - theta3, theta3))),
      fn     = log(ll);
```

is required that $q \geq 3$. A separate intercept must *not* be specified. The parameter vector is organized as follows:

$$\alpha_1 \ \ldots \ \alpha_{q-1} \ \beta_1 \ \ldots \ \beta_m$$

Box 3 Part of standard output from `qr2.cf` (logit model)

```
Model: ordinal logit.

Variables (cross-section)
-------------------------
Y   : Y
X1  : X1
X2  : X2

Categories of dependent variable.
Maximum number of categories: 100

Index               0         1         2         3   (Weighted)
Category            0         1         2         3  Observations
-------------------------------------------------------------------
Wave 1   N       5.00      5.00      4.00      6.00         20.00
         Pct    25.00     25.00     20.00     30.00

Maximum likelihood estimation.
Algorithm 5: Newton (I)

Number of model parameters: 5
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Convergence reached in 5 iterations.
Number of function evaluations: 6 (6,6)

Maximum of log likelihood: -24.4642
Norm of final gradient vector: 5.20097e-09
Last absolute change of function value: 1.43246e-10
Last relative change in parameters: 1.81865e-05

Idx Cat Term   Variable       Coeff      Error    C/Error  Signif
-----------------------------------------------------------------
  1   -  I 1   Alpha 1      -1.7466     1.3020    -1.3414  0.8202
  2   -  I 2   Alpha 2      -3.1531     1.4312    -2.2032  0.9724
  3   -  I 3   Alpha 3      -4.2253     1.5565    -2.7146  0.9934
  4   -  X     X1            1.0121     0.4637     2.1826  0.9709
  5   -  X     X2            1.3100     0.8949     1.4639  0.8568

Log likelihood (starting values):   -27.5245
Log likelihood (final estimates):   -24.4642
```

Box 4 Part of standard output from `qr2.cf` (probit model)

```
Model: ordinal probit.

Convergence reached in 5 iterations.
Number of function evaluations: 6 (6,6)

Maximum of log likelihood: -24.5741
Norm of final gradient vector: 5.01377e-10
Last absolute change of function value: 9.6685e-12
Last relative change in parameters: 3.42967e-06

Idx Cat Term   Variable       Coeff       Error     C/Error  Signif
-------------------------------------------------------------------
  1   -  I 1   Alpha 1      -1.0868      0.8207    -1.3243  0.8146
  2   -  I 2   Alpha 2      -1.9059      0.8692    -2.1927  0.9717
  3   -  I 3   Alpha 3      -2.5413      0.9193    -2.7645  0.9943
  4   -  X     X1            0.6122      0.2801     2.1853  0.9711
  5   -  X     X2            0.7563      0.5225     1.4475  0.8522

Log likelihood (starting values):   -29.5284
Log likelihood (final estimates):   -24.5741
```

To provide starting values, they must be given in this order. Default starting values are calculated as

$$\beta = 0 \qquad \text{and} \qquad \alpha_j = \log\left\{\frac{\tilde{\theta}_j}{1-\tilde{\theta}_j}\right\}$$

where $\tilde{\theta}_j$ is the weighted fraction of cases with $Y_i > R_j$. In most cases, this is sufficient to reach convergence.

Example 1 To illustrate estimation of ordered logit and probit models with TDA we replicate an example taken from the LIMDEP manual (version 6, p. 532). The data file is `qr2.dat`, shown in Box 1. The dependent variables has four categories. The command file, `qr2.cf` (Box 2), first uses two `qreg` commands to estimate an ordinal logit and an ordinal probit model, respectively. Finally, there is an `fml` command that illustrates how to estimate the ordinal logit model with a user-defined log-likelihood. Output from the `qreg` commands is shown in Box 3 (logit model) and 4 (probit model).

The `df` Option. Box 5 illustrates the `df` and `dtda` parameters using the data from Example 1 with a logit specification. The output file created with the `df` parameter contains the data used for model estimation and, in addition, estimated cumulative probabilities calculated according to formula (4). The column labelled `PROB` contains the value for the actually observed category.

Box 5 Illustration of `df` and `dtda` option

```
nvar(                          result of dtda parameter
  dfile = df,
  noc = 20,
  CaseID    [6.0] = c1 ,
  Wave      [2.0] = c2 ,
  Y         [10.4] = c3 ,
  X1        [10.4] = c4 ,
  X2        [10.4] = c5 ,
  PROB      [10.4] = c6 ,
  PROB0     [10.4] = c7 ,
  PROB1     [10.4] = c8 ,
  PROB2     [10.4] = c9 ,
  PROB3     [10.4] = c10,
);

 result of df paramter

 1  1  2.0000  1.1630  1.0000  0.1495  0.6771  0.3394  0.1495  0.0000
 2  1  2.0000  3.4530  0.0000  0.3251  0.8517  0.5846  0.3251  0.0000
 3  1  2.0000  3.7240  0.0000  0.3879  0.8831  0.6493  0.3879  0.0000
 4  1  0.0000  1.0900  0.0000  0.3445  0.3445  0.1141  0.0422  0.0000
 5  1  3.0000  3.8190  0.0000  0.0000  0.8927  0.6709  0.4109  0.0000
 6  1  1.0000  2.8380  1.0000  0.7368  0.9195  0.7368  0.4893  0.0000
 7  1  3.0000  1.5160  1.0000  0.0000  0.7498  0.4234  0.2009  0.0000
 8  1  3.0000  1.4170  1.0000  0.0000  0.7306  0.3992  0.1853  0.0000
 9  1  0.0000  2.4360  1.0000  0.8838  0.8838  0.6508  0.3894  0.0000
10  1  2.0000  3.3000  1.0000  0.6046  0.9480  0.8171  0.6046  0.0000
11  1  1.0000  1.0580  1.0000  0.3160  0.6535  0.3160  0.1365  0.0000
12  1  0.0000  2.6070  1.0000  0.9004  0.9004  0.6890  0.4313  0.0000
13  1  0.0000  2.1620  0.0000  0.6086  0.6086  0.2759  0.1154  0.0000
14  1  1.0000  2.1450  0.0000  0.2725  0.6045  0.2725  0.1136  0.0000
15  1  0.0000  1.1030  0.0000  0.3475  0.3475  0.1154  0.0427  0.0000
16  1  1.0000  1.9140  0.0000  0.2286  0.5475  0.2286  0.0921  0.0000
17  1  3.0000  3.6100  0.0000  0.0000  0.8707  0.6226  0.3609  0.0000
18  1  3.0000  3.5790  1.0000  0.0000  0.9603  0.8556  0.6698  0.0000
19  1  3.0000  3.3790  1.0000  0.0000  0.9518  0.8287  0.6235  0.0000
20  1  1.0000  3.1750  0.0000  0.5151  0.8126  0.5151  0.2666  0.0000
```

6.12.4 Multinomial Logit

We now consider a multinomial logit model for a depending variable with two or more unordered categories:

$$Y \in \{R_1, R_2, \ldots, R_q\} \qquad q \geq 2$$

Given a sample of $i = 1, \ldots, n$ individuals, observed values of Y will be denoted by y_i.

Although not essential, to describe the modeling approach we use the parlance of discrete choice models. It is assumed, then, that with each of the q alternatives $(R_1, \ldots, R_q)$ is associated a value of attractiveness, or utility, depending on characteristics of the alternative and on characteristics of the individual who perceives the alternatives. This may be written

$$u_j = u(x, z_j) + \epsilon_j \tag{1}$$

u_j is the attractiveness of the jth alternative assumed to depend on covariates x and z_j. The (row) vector x comprises covariates not varying with the alternatives; the (row) vector z_j comprises covariates specific for the jth alternative. ϵ_j is a random variable to account for stochastic influences.

The hypothesis of random utility maximization assumes that an individual chooses the jth alternative if its attractiveness is greater than the attractiveness of all other alternatives, that is if $u_j > u_k$ for all $k \neq j$.[1] Consequently, the condition for the jth alternative to be chosen is

$$u(x, z_j) + \epsilon_j > u(x, z_k) + \epsilon_k \qquad \text{for all} \quad k \neq j$$

This condition can be used to express the probability distribution of the dependent variable as follows:

$$\begin{aligned} P_j(x, z_j) &= \Pr\left(Y = R_j \mid x, z_j\right) \\ &= \Pr\left(\forall k \neq j : u(x, z_j) + \epsilon_j > u(x, z_k) + \epsilon_k\right) \end{aligned} \tag{2}$$

[1] The possibility that two or more alternatives have the same attractiveness is excluded. Through introduction of the random term in (1) it has a zero probability in the model. However, this may not be appropriate for all situations; for instance, if something is picked from a supermarket shelf. See Ullmann-Margalit and Morgenbesser [1977] for an interesting distinction between "picking" and "choosing".

These probabilities are defined as soon as the joint probability distribution for the random variables ϵ_j is specified. Let $f(\epsilon_1, \ldots, \epsilon_q)$ be their joint density function. Then

$$\Pr(\epsilon_1 \leq b_1, \ldots, \epsilon_q \leq b_q) = \int_{-\infty}^{b_1} \cdots \int_{-\infty}^{b_q} f(\epsilon_1, \ldots, \epsilon_q)\, d\epsilon_q \ldots d\epsilon_1$$

The probabilities $P_j(x, z_j)$ can be expressed in the same way by only changing the limits of integration according to the condition stated in (2). So one gets

$$P_j(x, z_j) = \int_{-\infty}^{b_1} \cdots \int_{-\infty}^{b_q} f(\epsilon_1, \ldots, \epsilon_q)\, d\epsilon_q \ldots d\epsilon_1 \tag{3}$$

$$b_k = \begin{cases} u(x, z_j) - u(x, z_k) + \epsilon_j & \text{if} \quad k \neq j \\ \infty & \text{if} \quad k = j \end{cases}$$

Based on this general choice model, a variety of different specifications is possible. Two things must be specified. First, one has to choose a distribution for the random vector $\epsilon = (\epsilon_1, \ldots, \epsilon_q)$. A standard choice is an extreme value distribution leading to a multinomial logit model. This is further discussed in the next section. Another choice would be the multivariate normal distribution leading to a probit model.

Secondly, one has to specify the dependence of the alternative's attractiveness on the covariates. For simplicity, all models discussed in this chapter assume a linear dependence, that is

$$u(x, z_j) = x\,\beta_j + z_j\,\gamma \tag{4}$$

Note that there is a separate parameter vector, β_j, for each category of the dependent variable, but only a single parameter vector, γ, for the alternative-specific covariates.

Logit Specification. The model is derived from the assumption that the components of the random vector $\epsilon = (\epsilon_1, \ldots, \epsilon_q)$ are independent and that each component follows a (type I) extreme value distribution. The density and distribution functions are, respectively

$$f_e(x) = \exp(-\exp(-x))\,\exp(-x) \tag{5}$$

$$F_e(x) = \exp(-\exp(-x))$$

The general model formulation given in (3) can be rewritten, then, as

$$P_j(x, z_j) = \int_{-\infty}^{\infty} f_e(\epsilon_j) \prod_{k \neq j} F_e(b_k)\, d\epsilon_j$$

Using the abbreviation $\tilde{u}_k = u(x, z_k)$ and the definition of the extreme value distribution in (5), gives

$$P_j(x, z_j) =$$

$$\int_{-\infty}^{\infty} \exp\{-\exp(-\epsilon_j)\} \exp(-\epsilon_j) \prod_{k \neq j} \exp\{-\exp(\tilde{u}_k - \tilde{u}_j - \epsilon_j)\}\, d\epsilon_j =$$

$$\int_{-\infty}^{\infty} \exp(-\epsilon_j) \prod_{k=1}^{q} \exp\{-\exp(\tilde{u}_k - \tilde{u}_j) \exp(-\epsilon_j)\}\, d\epsilon_j =$$

$$\int_{-\infty}^{\infty} \exp(-\epsilon_j) \exp\{-\exp(-u_j)\, C\}\, d\epsilon_j$$

with

$$C = \sum_{k=1}^{q} \exp(\tilde{u}_k - \tilde{u}_j)$$

The integral can be solved analytically. With $C' = \log(C)$, one finds that

$$\begin{aligned} P_j(x, z_j) &= \int_{-\infty}^{\infty} \exp\{-\epsilon_j - \exp(-\epsilon_j + C')\}\, d\epsilon_j \\ &= \frac{1}{C} \int_{-\infty}^{\infty} \exp\{-\epsilon_j + C' - \exp(-\epsilon_j + C')\}\, d\epsilon_j \end{aligned}$$

and, since by definition of the extreme value distribution function the last integral evaluates to unity, one gets

$$P_j(x, z_j) = \frac{\exp(\tilde{u}_j)}{\sum_{k=1}^{q} \exp(\tilde{u}_k)}$$

Finally, assuming a linear dependence on covariates as defined in (4), the model becomes

$$\Pr(Y = R_j \,|, x, z_j) = \frac{\exp(x\, \beta_j + z_j\, \gamma)}{\sum_{k=1}^{q} \exp(x\, \beta_k + z_k\, \gamma)} \tag{6}$$

Since not all of the parameter vectors β_j are estimable we add the restriction that $\beta_1 = 0$.

One should note that in the general model formulation given in (6) there is a different parameter vector, β_j, for each category of the dependent variable, but only a single parameter vector, γ, associated with the alternative-specific covariates. Therefore, all vectors z_j, for $j = 1, \ldots, q$, must have the same length, and this in turn is the length of the parameter vector γ. Of course, the model can be specified without any alternative-specific covariates leading to a standard multinomial logit model.

Maximum Likelihood Estimation. For a sample of $i = 1, \ldots, n$ observations, the likelihood of model (6) is

$$\mathcal{L} = \prod_{i=1}^{n} \prod_{j=1}^{q} \left[\frac{\exp(x_i \beta_j + z_{ij} \gamma)}{\sum_{k=1}^{q} \exp(x_i \beta_k + z_{ik} \gamma)} \right]^{w_{ij}}$$

with indicator variables w_{ij} defined by

$$w_{ij} = \begin{cases} 1 & \text{if} \quad y_i = R_j \\ 0 & \text{otherwise} \end{cases}$$

Since $\beta_1 = 0$, the expression can be simplified by defining

$$\tilde{z}_{ij} = z_{ij} - z_{i1}$$

The log-likelihood can then be written as

$$\ell = \sum_{i=1}^{n} \sum_{j=1}^{q} w_{ij} \log \left\{ \frac{\exp(x_i \beta_j + \tilde{z}_{ij} \gamma)}{1 + \sum_{k=2}^{q} \exp(x_i \beta_k + \tilde{z}_{ik} \gamma)} \right\}$$

Using the abbreviations

$$e_{ij} = \exp(x_i \beta_j + \tilde{z}_{ij} \gamma) \quad \text{and} \quad E_i = \sum_{k=2}^{q} \exp(x_i \beta_k + \tilde{z}_{ik} \gamma)$$

the derivatives are

$$\frac{\partial \ell}{\partial \beta_{r,s}} = \sum_{i=1}^{n}\sum_{j=1}^{q} w_{ij}\left\{\delta_{rj} - \frac{e_{ir}}{1+E_i}\right\} x_{i,s}$$

$$\frac{\partial \ell}{\partial \gamma_{s}} = \sum_{i=1}^{n}\sum_{j=1}^{q} w_{ij}\left\{\tilde{z}_{ij,s} - \frac{\sum_{k=2}^{q} e_{ik}\,\tilde{z}_{ik,s}}{1+E_i}\right\}$$

$$\frac{\partial^2 \ell}{\partial \beta_{r_1,s_1}\partial \beta_{r_2,s_2}} = \sum_{i=1}^{n}\sum_{j=1}^{q} w_{ij}\,\frac{e_{ir_1}}{1+E_i}\left\{\frac{e_{ir_2}}{1+E_i} - \delta_{r_1 r_2}\right\} x_{i,s_1}\,x_{i,s_2}$$

$$\frac{\partial^2 \ell}{\partial \gamma_{s_1}\partial \gamma_{s_2}} = \sum_{i=1}^{n}\sum_{j=1}^{q} w_{ij}\left\{\frac{\sum_{k=2}^{q} e_{ik}\,\tilde{z}_{ik,s_1}\sum_{k=2}^{q} e_{ik}\,\tilde{z}_{ik,s_2}}{(1+E_i)^2} - \frac{\sum_{k=2}^{q} e_{ik}\,\tilde{z}_{ik,s_1}\,\tilde{z}_{ik,s_2}}{1+E_i}\right\}$$

$$\frac{\partial^2 \ell}{\partial \beta_{r,s}\partial \gamma_{s_1}} = \sum_{i=1}^{n}\sum_{j=1}^{q} w_{ij}\,\frac{e_{ir}}{1+E_i}\left\{\frac{\sum_{k=2}^{q} e_{ik}\,\tilde{z}_{ik,s_1}}{1+E_i} - \tilde{z}_{ir,s_1}\right\} x_{i,s}$$

Implementation. The model can be estimated with the `qreg` command, using the model selection parameter `m=5`. In addition, one must use the `nq` parameter to specify the number of categories in the dependent variable.

The specification of covariates must follow the convention explained at the beginning of this chapter. A special convention is used for alternative-specific covariates. Each z-variable must be specified as a set of q variable names, q being the number of categories in the dependent variable. Each of these sets of variable name must be enclosed in brackets (see Example 2).

The number of model parameters is $m\,(q-1)+m'$. q is the number of categories of the dependent variable, m is the number of individual-specific covariates, and m' is the number of z-variables. The parameter vector (and accordingly the first and second derivatives of the log-likelihood) is organized in the following way:

$$\boxed{\beta_{21}\ \ldots\ \beta_{2m}\ \beta_{31}\ \ldots\ \beta_{3m}\ \beta_{q1}\ \ldots\ \beta_{qm}\ \gamma_1\ \ldots\ \gamma_q}$$

Box 1 Command file `qr3.cf`

```
nvar(
    dfile  = qr3.dat,
    X      = c1,
    Weight = c2,
    Y      = c3,
);
cwt = Weight;       use case weights
qreg(
   m = 5,           # select multinomial logit
   nq = 5,          # categories of dep variable
)= Y,X;

# same model with fml command

fml = xb2 = exp(beta20 + X * beta21),
      xb3 = exp(beta30 + X * beta31),
      xb4 = exp(beta40 + X * beta41),
      xb5 = exp(beta50 + X * beta51),
      xbb = 1 + xb2 + xb3 + xb4 + xb5,
      ll  = if (eq(Y,1),1 / xbb,
            if (eq(Y,2),xb2 / xbb,
            if (eq(Y,3),xb3 / xbb,
            if (eq(Y,4),xb4 / xbb, xb5 / xbb)))),
      fn  = Weight * log(ll);
```

β_{lj} is the parameter associated with the lth category and jth covariate ($l = 2, \ldots, q$, $j = 1, \ldots, m$). The first (lowest) category, R_1, is the reference. γ_k is the parameter associated with the kth set of alternative-specific covariates ($k = 1, \ldots, m'$).

The model is estimated by maximum likelihood as described above. All of TDA's maximization algorithms can be used. Default starting values are currently zero for all parameters. This may result in convergence problems. One should start, then, with simple models, successively adding covariates and using the results of simple models as starting values for more complex models.

Example 1 To illustrate estimation of multinomial logit models with TDA, our first example uses data taken from Dixon ([1990], p. 1048). The data file is `qr3.dat` (not shown, but contained in the example archive). The command, `qr3.cf`, is shown in Box 1. It contains the `qreg` command to estimate a multinomial logit model, and also a `fml` command that directly uses the corresponding log-likelihood. Part of the standard

Box 2 Part of standard output from `qr3.cf`

```
Model: multinomial logit.

Variables (cross-section)
-------------------------
Y   : Y
X1  : X

Categories of dependent variable.
Maximum number of categories: 100

Index              0       1       2       3       4   (Weighted)
Category           1       2       3       4       5  Observations
-----------------------------------------------------------------------
Wave 1   N     70.00  149.00  175.00  132.00   55.00        581.00
         Pct   12.05   25.65   30.12   22.72    9.47

Using case weights defined by: Weight

Maximum likelihood estimation.
Algorithm 5: Newton (I)

Convergence reached in 7 iterations.
Number of function evaluations: 8 (8,8)

Maximum of log likelihood: -826.748
Norm of final gradient vector: 6.22304e-12
Last absolute change of function value: 8.25065e-16
Last relative change in parameters: 2.47957e-07

Idx Cat Term   Variable       Coeff      Error    C/Error  Signif
-----------------------------------------------------------------
  1   2  I     Intercept     7.2428     1.3578     5.3342  1.0000
  2   2  X     X            -0.5035     0.1024    -4.9187  1.0000
-----------------------------------------------------------------
  3   3  I     Intercept     8.3752     1.3438     6.2326  1.0000
  4   3  X     X            -0.5842     0.1014    -5.7582  1.0000
-----------------------------------------------------------------
  5   4  I     Intercept    11.3662     1.3897     8.1788  1.0000
  6   4  X     X            -0.8723     0.1073    -8.1304  1.0000
-----------------------------------------------------------------
  7   5  I     Intercept    11.1873     1.5042     7.4373  1.0000
  8   5  X     X            -0.9375     0.1202    -7.7979  1.0000

Log likelihood (starting values):  -935.0834
Log likelihood (final estimates):  -826.7484
```

Box 3 Command file `qr4.cf`

```
nvar(
    dfile = qr4.dat,
    Z1    = c1,
    Z2    = c2,
    Z3    = c3,
    Y     = c4,
);
qreg(
    m = 5,                  # selection of logit model
    nq = 3,                 # number of categories
    ni = 1,                 # without an intercept
) = Y,(Z1,Z2,Z3);
```

output from the `qreg` command is shown in Box 2.

In this example, the number of categories in the dependent variable is $q = 5$ and there are no alternative-specific covariates. As indicated in the column labeled `Cat`, the output contains a separate block of estimated coefficients for each category, $j = 2, \ldots, q$.

Example 2 To illustrate the estimation of a multinomial logit models with alternative-specific covariates, we use example data from Daganzo [1979, p. 61].[2] The data are concerned with a choice between three different means of transportation. The data file, `qr4.dat`, contains four variables. `Y` is the dependent variable, with three categories, reflecting the actual choice. Variables `Z1`, `Z2`, and `Z3`, contain the expected transportation times associated with the three alternatives. The command file is shown in Box 3, part of its standard output in Box 4. In this example, there is only a single alternative-specific covariate and the model is estimated without an intercept. Since the dependent variable has three categories, the alternative-specific covariate must be specified by using three variable names, enclosed in brackets.

[2]The same data have been used for an illustration of KALOS, a program to estimate general multinomial logit models written by Röding, Küsters, and Arminger [1985]. However, there are minor differences in the data files. The TDA example archive contains the original Daganzo data as `qr4.dat` and the slightly different KALOS data set as `qr4a.dat`.

Box 4 Part of standard output from `qr4.cf`

```
Model: multinomial logit.

Variables (cross-section)
-------------------------
Y       : Y
Z1    1 : Z1
      2 : Z2
      3 : Z3

Model without intercept.
Checking available data (pmin=1)
Number of cases with valid data: 50

Categories of dependent variable.
Maximum number of categories: 100

Index               0         1         2   (Weighted)
Category            1         2         3  Observations
-------------------------------------------------------
Wave 1   N      14.00     29.00      7.00         50.00
         Pct    28.00     58.00     14.00

Maximum likelihood estimation.
Algorithm 5: Newton (I)

Number of model parameters: 1
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Convergence reached in 6 iterations.
Number of function evaluations: 7 (7,7)

Maximum of log likelihood: -33.3581
Norm of final gradient vector: 8.09432e-07
Last absolute change of function value: 4.59832e-09
Last relative change in parameters: 0.000120481

Idx Cat Term   Variable        Coeff      Error    C/Error  Signif
-----------------------------------------------------------------
  1   -  Z1    Z1            -0.3568     0.0776    -4.5966  1.0000

Log likelihood (starting values):   -54.9306
Log likelihood (final estimates):   -33.3581
```

Box 5 Illustration of `df` and `dtda` parameters

```
nvar(                              created by dtda parameter
  dfile = df,
  noc = 50,
  CaseID    [6.0] = c1 ,
  Wave      [2.0] = c2 ,
  Y         [10.4] = c3 ,
  Z1        [10.4] = c4 ,
  Z2        [10.4] = c5 ,
  Z3        [10.4] = c6 ,
  PROB      [10.4] = c7 ,
  PROB0     [10.4] = c8 ,
  PROB1     [10.4] = c9 ,
  PROB2     [10.4] = c10,
);

 first records of output file created with df parameter

 1  1  2.0000 16.4810 16.1960 23.8900  0.5083  0.4591  0.5083  0.0326
 2  1  2.0000 15.1230 11.3730 14.1820  0.6137  0.1610  0.6137  0.2252
 3  1  2.0000 19.4690  8.8220 20.8190  0.9651  0.0216  0.9651  0.0133
 4  1  2.0000 18.8470 15.6490 21.2800  0.6880  0.2198  0.6880  0.0922
 5  1  2.0000 12.5780 10.6710 18.3350  0.6364  0.3223  0.6364  0.0413
 6  1  1.0000 11.5130 20.5820 27.8380  0.9595  0.9595  0.0377  0.0028
 7  1  1.0000 10.6510 15.5370 17.4180  0.7910  0.7910  0.1383  0.0707
 8  1  1.0000  8.3590 15.6750 21.0500  0.9223  0.9223  0.0678  0.0100
```

The `df` Option. Box 5 illustrates the `df` and `dtda` parameters using the data from Example 2. The output file created with the `df` parameter contains the data used for model estimation and, in addition, estimated probabilities calculated according to formula (6). The column labelled `PROB` contains the value for the actually observed category.

6.12.5 Multivariate Probit

We now consider a probit specification of the general model that was decribed in 6.12.4. For an extensive discussion see Daganzo [1979]. $Y \in \{R_1, \ldots, R_q\}$ is the dependent variable. The attractiveness of category R_j is

$$u_j = x\beta_j + z_j\gamma$$

depending on covariate vectors x and z_j. The probability for alternative R_j may then be written as

$$\begin{aligned} P_j(x, z_j) &= \Pr(Y = R_j \mid x, z_j) \\ &= \Pr(\forall k \neq j : u_j + \epsilon_j > u_k + \epsilon_k) \\ &= \Pr(\forall k \neq j : \epsilon_k - \epsilon_j < u_j - u_k) \end{aligned} \tag{1}$$

The assumption is that

$$\epsilon = (\epsilon_1, \ldots, \epsilon_q) \sim \mathcal{N}(0, \Sigma_q)$$

meaning that ϵ follows a q-dimensional normal distribution with zero mean and covariance matrix Σ_q. One dimension can be saved by using

$$\epsilon_k^{(j)} := \epsilon_k - \epsilon_j, \quad u_k^{(j)} := u_j - u_k$$

We may then write

$$P_j(x, z_j) = \Pr(\forall k \neq j : \epsilon_k^{(j)} < u_k^{(j)}) \tag{2}$$

Let $\epsilon_{(j)}$ denote the $(q-1)$-vector consisting of $\epsilon_1^{(j)}, \ldots, \epsilon_q^{(j)}$ for $k \neq j$. There is then a simple linear transformation

$$\epsilon^{(j)} = T^{(j)}@, \epsilon$$

where the $(q-1, q)$-matrix $T^{(j)}$ is defined by

$$T_{tt'}^{(j)} = \begin{cases} -1 & \text{if } t' = j \\ 1 & \text{if } t' < j \text{ and } t' = t \\ 1 & \text{if } t' > j \text{ and } t' = t+1 \\ 0 & \text{otherwise} \end{cases}$$

Then, see, e.g., Tong [1990, p. 26], $\epsilon^{(j)}$ follows a $(q-1)$-dimensional normal distribution:

$$\epsilon^{(j)} \sim \mathcal{N}\big(0, T^{(j)}\Sigma_q T^{(j)\prime}@,\big)$$

Finally, let $u_{(j)}$ denote the $(q-1)$-vector consisting of $u_1^{(j)}, \ldots, u_q^{(j)}$ for $k \neq j$ and $\Phi_{q-1}^{(j)}$ the distribution function of a $(q-1)$-dimensional normal distribution with zero mean and covariance matrix

$$\Sigma_{q-1}^{(j)} := T^{(j)}\Sigma_q T^{(j)\prime}$$

we can simply write

$$P_j(x, z_j) = \Phi_{q-1}^{(j)}(u^{(j)}) \tag{3}$$

Maximum Likelihood Estimation. Now assume a sample of observations, (y_i, x_i, z_{ij}), for $i = 1, \ldots, n$, where the covariate vector x has m components. Setting $\beta_1 = 0$, the model parameters are $\beta_2, \ldots, \beta_m$, γ. Using (3), the log-likelihood can be written as

$$\ell(\beta_2, \ldots, \beta_m, \gamma) = \sum_{i=1}^{n} \log\left(\Phi_{q-1}^{(y_i)}(u^{(y_i)})\right) \tag{4}$$

Implementation. The multivariate probit model can be estimated with the `qreg` command, the model selection parameter is `m=6`. The specification of variables on the right-hand side follows the same convention as was explained for the multinomial logit model in **6.12.4**. If there are two or more panel waves, the data are pooled over waves. The covariance matrix

$$\Sigma_q = \big(\sigma_{ij}\big)$$

is assumed to be a correlation matrix ($\sigma_{ii} = 1$), and all elements in the lower triangle are possible model parameters. The parameter vector is organized as

$$\beta_{21} \ldots \beta_{2m} \ \ldots \ \beta_{q1} \ldots \beta_{qm} \ \gamma_1 \ \ldots \ \gamma_q \ \sigma_{21} \ \sigma_{31} \ \sigma_{32} \ \ldots \ \sigma_{q1} \ \ldots \ \sigma_{q,q-1}$$

Of course, most often the model will not be estimable without further constraints on the components of the correlation matrix. This can be done with the `con` parameter for parameter constraints, see **5.6.6**.

Optional Parameterizations. The off-diagonal elements of the correlation matrix must satisfy $-1 < \sigma_{ij} < 1$. TDA's standard function minimization routines cannot account for this constraint, and so it can easily happen that the estimation procedure cannot continue because of a violation of this constraint. For this reason, TDA offers three different parameterizations that can be selected with the `opt` parameter. Let σ^*_{ij} denote the components in the parameter vector corresponding to the lower triangle of the correlation matrix.

1. If `opt = 1` (default), the correlation matrix for model estimation is created by setting $\sigma_{ij} = \sigma^*_{ij}$, without any further reparameterization.

2. If `opt = 2`, the transformation

$$\sigma_{ij} = 2\frac{\exp(\sigma^*_{ij})}{1+\exp(\sigma^*_{ij})} - 1$$

is applied before creating the covariance from the σ_{ij} parameters.

3. If `opt = 3`, it is assumed that the parameters σ^*_{ij} are the elements of a Cholesky decomposition of Σ_q (see, e.g., Tong [1990, p. 184]):

$$\Sigma_q = CC' \quad \text{where} \quad C = \begin{bmatrix} 1 & & & & \\ \sigma^*_{21} & 1 & & & \\ \sigma^*_{31} & \sigma^*_{32} & 1 & & \\ \vdots & \vdots & \vdots & & \\ \sigma^*_{q1} & \sigma^*_{q2} & \sigma^*_{q,q-1} & \cdots & 1 \end{bmatrix}$$

In any case, the output file that can be requested with the `ppar` parameter will contain the estimated correlation matrix, Σ_q. It is easy, therefore, to re-estimate the model with `opt=1`, given that one has achieved results with `opt=2` or `opt=3`.

Numerical Integration. The main difficulty is the evaluation of the distribution function, $\Phi^{(j)}_{q-1}$, of the $(q-1)$-dimensional normal distribution. Different approaches have been proposed for the evaluation of the corresponding $(q-1)$-dimensional integral. For the current implementation, TDA uses a numerical integration procedure developed by Drezner [1992], based on a Gauss quadrature method. The procedure is computationally very intensive, and so it might not be possible to estimate models with a large number of categories in the dependent variable. The

Box 1 Command file `qr6.cf`

```
nvar(
    dfile  = qr3.dat,
    X      = c1,
    Weight = c2,
    Y      = c3,
);
cwt = Weight;        use case weights
qreg(
    m = 6,           # select multivariate probit
    nq = 5,          # categories of dep variable
    ppar = par,      # write estimated parameters and corr. matrix
    con = b9 = 0,    # use identity correlation matrix
    con = b10 = 0,
    con = b11 = 0,
    con = b12 = 0,
    con = b13 = 0,
    con = b14 = 0,
    con = b15 = 0,
    con = b16 = 0,
    con = b17 = 0,
    con = b18 = 0,
)= Y,X;
```

advantage is, however, that the numerical accuracy of results can be controlled. There are two parameters, `nhp` and `eps`.

1. The `nhp =` k parameter can be used to specify the number of integration points, k, in the Gauss quadrature procedure. If $2 \leq k \leq 10$, the algorithm performs one Gauss quadrature with k points for each dimension of the integral. Default is `nhp = 6`.

2. If $12 \leq k \leq 20$ the algorithm performs Gauss quadratures with $k-10$ points progressively until the accuracy specified with the `eps` parameter is reached. Default is `eps =` 10^{-6}. Note that the value of this parameter is only used when $12 \leq k \leq 20$.

The algorithm may not be able to achieve the required accuracy. This will not be regarded as an error, but the standard output will then contain a warning message. Note that the same algorithm is also used for the `mvn` operator, see 5.2.5.7. This operator can be used to become familiar with the integration procedure and to evaulate the distribution function for arbitrary input values.

Box 2 Part of standard output from `qr6.cf`

```
Maximum likelihood estimation.
Algorithm 8: CES (tensor model [0])

Number of model parameters: 18
Type of covariance matrix: 2
Maximum number of iterations: 200
Convergence criterion: 6
Tolerance for scaled gradient: 1e-05
Tolerance for scaled parameter change: 1e-08

Control of integration (nhp): 6
Type of parameterization: 1
Protocol will be written to: p

Convergence reached in 5 iterations.
Number of function evaluations: 321 (0,0)

Maximum of log likelihood: -828.232
Norm of final gradient vector: 0.0031299
Final scaled gradient: 2.71293e-06
Final scaled parameter change: 0.00511032
```

Idx	Cat	Term	Variable	Coeff	Error	C/Error	Signif
1	2	I	Intercept	2.8003	0.5530	5.0641	1.0000
2	2	X	X	-0.1915	0.0431	-4.4467	1.0000
3	3	I	Intercept	3.3703	0.5456	6.1771	1.0000
4	3	X	X	-0.2320	0.0426	-5.4507	1.0000
5	4	I	Intercept	4.8489	0.5610	8.6428	1.0000
6	4	X	X	-0.3740	0.0445	-8.3974	1.0000
7	5	I	Intercept	4.5686	0.6055	7.5450	1.0000
8	5	X	X	-0.3867	0.0493	-7.8409	1.0000
9	-	S	Sigma 2, 1	0.0000	0.0000	---	---
10	-	S	Sigma 3, 1	0.0000	0.0000	---	---
11	-	S	Sigma 3, 2	0.0000	0.0000	---	---
12	-	S	Sigma 4, 1	0.0000	0.0000	---	---
13	-	S	Sigma 4, 2	0.0000	0.0000	---	---
14	-	S	Sigma 4, 3	0.0000	0.0000	---	---
15	-	S	Sigma 5, 1	0.0000	0.0000	---	---
16	-	S	Sigma 5, 2	0.0000	0.0000	---	---
17	-	S	Sigma 5, 3	0.0000	0.0000	---	---
18	-	S	Sigma 5, 4	0.0000	0.0000	---	---

```
Log likelihood (starting values):  -935.1167
Log likelihood (final estimates):  -828.2320
```

Box 3 Command file `qr7.cf`

```
nvar(
    dfile = qr4.dat,
    Z1    = c1,
    Z2    = c2,
    Z3    = c3,
    Y     = c4,
);
qreg(
    m = 6,                  # model selection
    nq = 3,                 # number of categories
    ni = 1,                 # without intercept
    df = df,                # print data and estimated probabilities
    dtda = t,               # description file
    xp = -0.1716,0,0,0,     # starting values taken from model with
                            # zero correlations
    con = b3 = 0,           # keep only sigma(2,1)
    con = b4 = 0,
    ppar = par,             # print parameter and corr. matrix
    df = df,                # write data and estimated probabilities
    dtda = t,
) = Y,(Z1,Z2,Z3);
```

Available Minimization Algorithms. While it would be possible to use the same integration procedure also for an evaluation of derivatives of the log-likelihood, this has not been implemented in TDA. Only function values, no derivataives of the likelihood function are available. As a consequence, the multivariate probit model can only be estimated with one of the algorithms, `mina=7` or `mina=8` (see 5.6.2). Default is `mina=8`. Both algorithms numerically approximate first and second derivatives. A numerical approximation to the Hessian of the log-likelihood is also used to estimate a covariance matrix for the model parameters (consequently, always `ccov = 2`).

Starting Values. As default, all parameters have a zero initial value. This will often not lead to convergence. So one should specify suitable starting values with the `xp` or `dsv` parameter. Note that an output file written with the `ppar` parameter can be used as an input file with the `dsv` parameter. Of course, file names should be different when using both parameters simultaneously.

Example 1 As a first illustration we replicate multinomial logit model from the first example in 6.12.4, now with a multivariate probit speci-

Box 4 Part of standard output from `qr7.cf`

```
Maximum likelihood estimation.
Algorithm 8: CES (tensor model [0])

Number of model parameters: 4
Type of covariance matrix: 2
Maximum number of iterations: 200
Convergence criterion: 6
Tolerance for scaled gradient: 1e-05
Tolerance for scaled parameter change: 1e-08
Scaling factor: -1

Control of integration (nhp): 6
Type of parameterization: 1
Got 4 starting value(s) from xp parameter.
Number of constraints: 2

Convergence reached in 4 iterations.
Number of function evaluations: 40 (0,0)

Maximum of log likelihood: -34.4166
Norm of final gradient vector: 2.21297e-06
Final scaled gradient: 5.87576e-08
Final scaled parameter change: 0.000629914

Idx Cat Term   Variable        Coeff      Error    C/Error  Signif
-------------------------------------------------------------------
  1   -  Z1    Z1            -0.1787     0.0349    -5.1178  1.0000
-------------------------------------------------------------------
  2   -  S     Sigma 2, 1     0.4134     0.6320     0.6542  0.4870
  3   -  S     Sigma 3, 1     0.0000     0.0000        ---     ---
  4   -  S     Sigma 3, 2     0.0000     0.0000        ---     ---

Log likelihood (starting values):   -34.5825
Log likelihood (final estimates):   -34.4166
```

fication. Box 1 shows the command file, `qr6.cf`. In this example, as is often a good starting point for the estimation of a multivariate probit model, the constraints assure that all correlations are zero.

Although the command files relies on default starting values convergence has been achieved in 5 iterations. However, due to the fact that the minimization algorithm numerically approximates first and second derivatives of the log-likelihood function, it needs altogether 321 evaluations of the likelihood function. Multiplying this with the number of cases shows how often the multivariate normal integral has been evaluated.

6.14 Models for Count Data

We consider a dependent variable, Y, with non-negative integer values. Values of Y may be interpreted as the number of times that some event occurs. Standard regression models for this type of dependent variable include the Poisson model and several variants of a negative binomial model. While TDA does not offer a specific command to estimate these kinds of models, estimation is actually quite easy when using the `fml` command and the operators for the Poisson and the negative binomial distribution. This chapter shortly explains how to set up the likelihood for some standard models and gives some examples. For a discussion of the `fml` command see **6.11.2**.

6.14.1 Poisson Regression

The most simple model is based on the assumption that Y follows a Poisson distribution, that is,

$$\Pr(Y = k) = \frac{\theta^k}{k!} \exp(-\theta) \qquad \theta > 0,\ k = 0, 1, 2, \ldots$$

This implies that $\mathrm{E}(Y) = \mathrm{Var}(Y) = \theta$. The standard Poisson regression model uses the parameterization

$$\theta = \exp(x\beta)$$

where x denotes a (row) vector of covariates. For numerical evaluation, TDA offers the `poisson`

$$\texttt{poisson}(\theta, k) \equiv \log\left(\frac{\theta^k}{k!} \exp(-\theta)\right)$$

This is a type I operator and allows for automatic differentiation with respect to its first argument, θ. It is therefore quite easy to estimate a Poisson regression model with the `fml` command. Given a sample of observations, (y_i, x_i), for $i = 1, \ldots, n$, the log-likelihood is simply

$$\ell = \sum_{i=1}^{n} \texttt{poisson}(\exp(x_i\beta), y_i)$$

Example 1 To illustrate model estimation we use the ship damage data, data file `cd1.dat`, shown in Box 1. (These data have been used by many authors. See, e.g., McCullagh and Nelder [1983, p. 137], Greene [1992, p. 546], Winkelmann and Zimmermann [1991].) Box 2 shows the command file, `cd1.cf`. The `nvar` command creates the necessary variables based on data file `cd1.dat`. Then follows the `fml` command. Since the likelihood of the Poisson model is well behaved, we do not need any parameters. It suffices to formulate the expression for the log-likelihood on the right-hand side of the command. This is done in three steps. The first step creates the intermediate expression `xb`, corresponding to $x\beta$. Note that parameter names can be arbitrary strings consisting of lower

Box 1 Ship damage data file `cd1.dat` (McCullagh and Nelder [1983, p. 137]

```
# Col  1     : number of damage incidents
# Col  2     : aggregate months of service
# Col  3 - 6 : dummy variables for ship type (A - E)
# Col  7 - 9 : dummy variables for year of construction
# Col 10     : dummy variables for period of operation

 0     127  0  0  0  0  0  0  0  0
 0      63  0  0  0  0  0  0  0  1
 3    1095  0  0  0  0  1  0  0  0
 4    1095  0  0  0  0  1  0  0  1
 6    1512  0  0  0  0  0  1  0  0
18    3353  0  0  0  0  0  1  0  1
-1       0  0  0  0  0  0  0  1  0
11    2244  0  0  0  0  0  0  1  1
39   44882  1  0  0  0  0  0  0  0
29   17176  1  0  0  0  0  0  0  1
58   28609  1  0  0  0  1  0  0  0
53   20370  1  0  0  0  1  0  0  1
12    7064  1  0  0  0  0  1  0  0
44   13099  1  0  0  0  0  1  0  1
-1       0  1  0  0  0  0  0  1  0
18    7117  1  0  0  0  0  0  1  1
 1    1179  0  1  0  0  0  0  0  0
 1     552  0  1  0  0  0  0  0  1
 0     781  0  1  0  0  1  0  0  0
 1     676  0  1  0  0  1  0  0  1
 6     783  0  1  0  0  0  1  0  0
 2    1948  0  1  0  0  0  1  0  1
-1       0  0  1  0  0  0  0  1  0
 1     274  0  1  0  0  0  0  1  1
 0     251  0  0  1  0  0  0  0  0
 0     105  0  0  1  0  0  0  0  1
 0     288  0  0  1  0  1  0  0  0
 0     192  0  0  1  0  1  0  0  1
 2     349  0  0  1  0  0  1  0  0
11    1208  0  0  1  0  0  1  0  1
-1       0  0  0  1  0  0  0  1  0
 4    2051  0  0  1  0  0  0  1  1
 0      45  0  0  0  1  0  0  0  0
-1       0  0  0  0  1  0  0  0  1
 7     789  0  0  0  1  1  0  0  0
 7     437  0  0  0  1  1  0  0  1
 5    1157  0  0  0  1  0  1  0  0
12    2161  0  0  0  1  0  1  0  1
-1       0  0  0  0  1  0  0  1  0
 1     542  0  0  0  1  0  0  1  1
```

Box 2 Command file `cd1.cf`

```
nvar(
    dfile = cd1.dat,
    isel = ge(c1,0),    # select valid cases
    NDI     = c1,
    Service = c2,
    B       = c3,
    C       = c4,
    D       = c5,
    E       = c6,
    C60     = c7,
    C65     = c8,
    C70     = c9,
    P75     = c10,
    LOGS    = log(Service),
);
fml(    # we don't need any parameters

) = xb  = b0 + B * bb + C * bc + D * bd + E * be +
C60 * bc60 + C65 * bc65 + C70 * bc70 + P75 * bp75 + LOGS * blogs,
exb = exp(xb),
fn  = poisson(exb,NDI);
```

case letters and digits, but must begin with a lower case letter. The second step creates the intermediate expression `exb` $\equiv \exp(x\beta)$. The final step uses the `poisson` operator to formulate the log-likelihood. This final expression is automatically summed over all cases in the currently active data matrix.

Box 3 shows part of the standard output. Since the `fml` command uses a default set up, the minimization algorithm is TDA' default Newton I algorithm, and all starting values are zero. Nevertheless, convergence is achieved with 12 function calls. Since the `poisson` operator allows for automatic differentiation (in its first argument), first and second derivatives of the log-likelihood are automatically calculated, and the second derivatives are used for the standard errors (`ccov = 2`).

Example 2 As noted by McCullagh and Nelder [1983, p. 138], one can assume that the parameter for the logarithm of aggregate months service (`blogs`) should be 1. This constraint can easily be formulated by adding

```
con = b9 = 1,
```

to the `fml` command (see command file `cd1a.cf` in the example archive). Note, however, that one cannot use the (arbitrary) name of the param-

Box 3 Part of standard output from `cd1.cf` (Poisson regression)

```
ML estimation of user-defined model.
Algorithm 5: Newton (I)

Number of model parameters: 10
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Idx  Parameter   Starting value
  1  b0           0.00000000e+00
  2  bb           0.00000000e+00
  3  bc           0.00000000e+00
  4  bc60         0.00000000e+00
  5  bc65         0.00000000e+00
  6  bc70         0.00000000e+00
  7  bd           0.00000000e+00
  8  be           0.00000000e+00
  9  blogs        0.00000000e+00
 10  bp75         0.00000000e+00

Convergence reached in 8 iterations.
Number of function evaluations: 12 (12,12)

Maximum of log likelihood: -67.8354
Norm of final gradient vector: 2.84694e-11
Last absolute change of function value: 5.57244e-14
Last relative change in parameters: 2.05576e-06

Idx  Parameter       Value      Error    Value/E  Signif
------------------------------------------------------------
  1  b0            -5.5940     0.8724    -6.4121  1.0000
  2  bb            -0.3499     0.2702    -1.2948  0.8046
  3  bc            -0.7631     0.3382    -2.2565  0.9760
  4  bc60           0.6625     0.1536     4.3124  1.0000
  5  bc65           0.7597     0.1777     4.2763  1.0000
  6  bc70           0.3697     0.2458     1.5040  0.8674
  7  bd            -0.1355     0.2971    -0.4560  0.3516
  8  be             0.2739     0.2418     1.1331  0.7428
  9  blogs          0.9027     0.1018     8.8672  1.0000
 10  bp75           0.3703     0.1181     3.1339  0.9983

Log likelihood (starting values):  -870.2939
Log likelihood (final estimates):   -67.8354
```

Box 4 Part of standard output from `qr1a.cf`

```
Idx  Parameter        Value       Error     Value/E  Signif
-------------------------------------------------------------
  1  b0             -6.4059      0.2174   -29.4600  1.0000
  2  bb             -0.5433      0.1776    -3.0595  0.9978
  3  bc             -0.6874      0.3290    -2.0891  0.9633
  4  bc60            0.6971      0.1496     4.6587  1.0000
  5  bc65            0.8184      0.1698     4.8207  1.0000
  6  bc70            0.4534      0.2332     1.9446  0.9482
  7  bd             -0.0760      0.2906    -0.2614  0.2062
  8  be              0.3256      0.2359     1.3803  0.8325
  9  blogs           1.0000         ---        ---     ---
 10  bp75            0.3845      0.1183     3.2507  0.9988

Log likelihood (starting values): -161157.7779
Log likelihood (final estimates):     -68.2808
```

eter in the formulation of constraints but has to use a reference to the number of the parameter. In this case, `blogs` is the 9th parameter, and is therefore referred to by `b9`. Estimation results are shown in Box 4.

6.14.2 Compound Poisson Models

The Poisson distribution is obviously restrictive in the assumption that the mean equals the variance. Several less restrictive distributions have been proposed in the literature. A standard approach thinks in terms of a compound Poisson distribution. It is assumed, then, that the mean of the Poisson distribution, θ, is a random variable. Let $f_\theta(\theta;\theta^*)$ denote its density function, depending on some further parameter vector, θ^*. A compound Poisson distribution is then given by

$$\Pr(Y=k) \;=\; \int_0^\infty \frac{\theta^k}{k!}\exp(-\theta)\, f_\theta(\theta;\theta^*)\, d\theta$$

Following the discussion in Cameron and Trivedi [1986], one arrives at a relatively simple model when a gamma mixture distribution ist use, that is,

$$f_\theta(\theta;\alpha,\gamma) = \frac{1}{\Gamma(\alpha)}\,\frac{1}{\theta}\left(\frac{\alpha\theta}{\gamma}\right)^\alpha \exp\left\{\frac{-\alpha\theta}{\gamma}\right\}$$

having mean $\mathrm{E}(\theta)=\gamma$ and variance $\mathrm{Var}(\theta)=\gamma^2/\alpha$. The distribution for Y then becomes

$$\Pr(Y=k) \;=\; \frac{(\alpha/\gamma)^\alpha}{k!\,\Gamma(\alpha)}\int_0^\infty \theta^{\alpha+k-1}\exp\left\{-\theta(\frac{\alpha}{\gamma}+1)\right\} d\theta$$

Making the substitution $\theta \to \delta\gamma/(\alpha+\gamma)$, we find

$$\Pr(Y=k) \;=\; \frac{(\alpha/\gamma)^\alpha}{k!\,\Gamma(\alpha)}\left(\frac{\gamma}{\alpha+\gamma}\right)^{\alpha+1}\int_0^\infty \delta^{\alpha+k-1}\exp(-\delta)\, d\theta$$

The integral (Euler's integral, see Abramowitz and Stegun [1964, p. 255]) equals $\Gamma(\alpha+k)$, so we find

$$\Pr(Y=k) \;=\; \frac{\Gamma(\alpha+k)}{\Gamma(k+1)\,\Gamma(\alpha)}\left(\frac{\gamma}{\alpha+\gamma}\right)^k\left(\frac{\alpha}{\alpha+\gamma}\right)^\alpha \qquad (1)$$

This is called a (general) negative binomial distribution. The distribution has two parameters, α and γ. The mean is $\mathrm{E}(Y)=\gamma$, the variance is $\mathrm{Var}(Y)=\gamma+\gamma^2/\alpha$.

To arrive at a regression model, one can make both parameters dependent on covariates. A particularly convenient approach has been proposed by Cameron and Trivedi [1986]:

$$\begin{aligned}\gamma &= \exp(x\beta)\\ \alpha &= \frac{1}{\sigma}\exp(x\beta)^q\end{aligned}$$

where q is some given constant and σ is a model parameter to be estimated. The mean of Y is directly given by $\exp(x\beta)$, and the variance is

$$\mathrm{Var}(Y) = \mathrm{E}(Y) + \sigma\,\mathrm{E}(Y)^{2-q}$$

In particular, if $q = 1$, one gets

$$\mathrm{Var}(Y) = \mathrm{E}(Y) + \sigma\,\mathrm{E}(Y)$$

called NEGBIN I model by Cameron and Trivedi. If $q = 0$, one gets

$$\mathrm{Var}(Y) = \mathrm{E}(Y)\big(1 + \sigma\,\mathrm{E}(Y)\big)$$

called NEGBIN II model by Cameron and Trivedi.[1] In both cases, if $\sigma \to 0$, one arrives at the standard Poisson regression model.

The `negbin` Operator In order to evaluate the negative binomial distribution one can use TDA's `negbin` operator, with syntax

$$\texttt{negin}(\alpha, \gamma, k) \equiv \log\left(\frac{(\Gamma(\alpha+k)}{\Gamma(k+1)\,\Gamma(\alpha)}\left(\frac{\gamma}{\alpha+\gamma}\right)^k\left(\frac{\alpha}{\alpha+\gamma}\right)^\alpha\right)$$

that directly evaluates the logarithm of the distribution as given in (1). This is a type I operator and allows for automatic differentiation with respect to its first two arguments, α and γ. The derivatives are calculated as follows.

[1] This is the parameterization of the negative binomial model used in LIMDEP, see Greene [1992, p. 539].

$$\begin{aligned}
\frac{\partial \texttt{negbin}(\alpha,\gamma,k)}{\partial\alpha} &= \texttt{digam}(\alpha+k)-\texttt{digam}(\alpha)+\\
&\qquad \log\left(\frac{\alpha}{\alpha+\gamma}\right)+\frac{\gamma-k}{\alpha+\gamma}\\
\frac{\partial \texttt{negbin}(\alpha,\gamma,k)}{\partial\gamma} &= \frac{\alpha}{\alpha+\gamma}\left[\frac{k}{\gamma}-1\right]\\
\frac{\partial^2 \texttt{negbin}(\alpha,\gamma,k)}{\partial\alpha\partial\alpha} &= \texttt{trigam}(\alpha+k)-\texttt{trigam}(\alpha)+\\
&\qquad \frac{1}{\alpha+\gamma}\left[\frac{\gamma}{\alpha}-\frac{\gamma-k}{\alpha+\gamma}\right]\\
\frac{\partial^2 \texttt{negbin}(\alpha,\gamma,k)}{\partial\gamma\partial\gamma} &= \frac{\alpha}{\alpha+\gamma}\left[\frac{1-k/\gamma}{\alpha+\gamma}-\frac{k}{\gamma^2}\right]\\
\frac{\partial^2 \texttt{negbin}(\alpha,\gamma,k)}{\partial\alpha\partial\gamma} &= \frac{k-\gamma}{(\alpha+\gamma)^2}
\end{aligned}$$

`digam` and `trigam` are operators that evaluate, respectively, the first and second derivative of the logarithm of the gamma function.

The `negbin` operator allows an easy formulation of the log-likelihood for a regression model with a negative binomial distribution. Given a sample of observations (y_i, x_i), for $i = 1,\ldots,n$, the log-likelihood is simply

$$\ell = \sum_{i=1}^{n} \texttt{negbin}\,(\alpha, \exp(x_i\beta), y_i)$$

where α can be made in an arbitrary way a function of the mean, i.e., $\exp(x_i\beta)$.

Example 1 To illustrate, we estimate a NEGBIN II model for the ship damage data that were used in 6.14.1. Box 1 shows part of the command file, `cd2.cf` (the first part is identical with the `nvar` command in command file `cd1.cf`.) The expression

```
sigma = exp(sig),
```

is used to assure that σ is always positive. The model is estimated with algorithm 8 that is somewhat more robust than TDA's standard algorithm for function minimization.

Box 1 Part of command file `cd2.cf`

```
fml(
    mina = 8,       # select somewhat more robust algorithm

) = xb  = b0 + B * bb + C * bc + D * bd + E * be +
  C60 * bc60 + C65 * bc65 + C70 * bc70 + P75 * bp75 + LOGS * blogs,
  gamma = exp(xb),
  sigma = exp(sig),
  alpha = 1 / sigma,
  fn  = negbin(alpha,gamma,NDI);
```

Box 2 shows the estimation results. σ is almost zero, indicating that the NEGBIN II model does not give a better fit than the standard Poisson model. In fact, the parameter estimates are almost the same as those shown in Box 3. Of course, the huge standard error for the `sig` parameter indicates that the estimate is not reliable.

Box 2 Part of standard output from `cd2.cf`

```
Maximum likelihood estimation.
Algorithm 8: CES (tensor model [2])

Number of model parameters: 11
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 6
Tolerance for scaled gradient: 1e-05
Tolerance for scaled parameter change: 1e-08
Scaling factor: -1

Idx  Parameter   Starting value
  1  b0           0.00000000e+00
  2  bb           0.00000000e+00
  3  bc           0.00000000e+00
  4  bc60         0.00000000e+00
  5  bc65         0.00000000e+00
  6  bc70         0.00000000e+00
  7  bd           0.00000000e+00
  8  be           0.00000000e+00
  9  blogs        0.00000000e+00
 10  bp75         0.00000000e+00
 11  sig          0.00000000e+00

Convergence reached in 17 iterations.
Number of function evaluations: 52 (19,19)

Maximum of log likelihood: -67.8354
Norm of final gradient vector: 5.25902e-05
Final scaled gradient: 4.90683e-06
Final scaled parameter change: 0.066398

Idx  Parameter       Value      Error    Value/E  Signif
-------------------------------------------------------
  1  b0            -5.5940     0.8724    -6.4121  1.0000
  2  bb            -0.3499     0.2702    -1.2948  0.8046
  3  bc            -0.7631     0.3382    -2.2565  0.9760
  4  bc60           0.6625     0.1536     4.3124  1.0000
  5  bc65           0.7597     0.1777     4.2763  1.0000
  6  bc70           0.3697     0.2458     1.5040  0.8674
  7  bd            -0.1355     0.2971    -0.4560  0.3516
  8  be             0.2739     0.2418     1.1331  0.7428
  9  blogs          0.9027     0.1018     8.8672  1.0000
 10  bp75           0.3703     0.1181     3.1339  0.9983
 11  sig          -15.0643   213.8136    -0.0705  0.0562

Log likelihood (starting values):  -270.3274
Log likelihood (final estimates):   -67.8354
```

6.15 Generalized Linear Models

This chapter deals with generalized linear models. It contains the following sections.

6.15.1 Introduction

Let Y denote a single dependent variable and (y_i, x_i), for $i = 1, \ldots, n$, a sample of observations. For case i, y_i is the value of the dependent variable, and x_i is a vector of covariates. To find a model for the distribution of Y, conditional on the information in covariates, one can assume a parametric density for Y, say $f(y; \theta)$, and then make the parameter vector θ dependent on x. In general terms, this approach might be written as

$$f(y; \theta) \quad \text{with} \quad \theta = u(x, \beta) \tag{1}$$

where the function $u(x, \beta)$ depends on information given by the covariate vector x, and a new parameter vector β. The log-likelihood for model estimation may then be written as

$$\ell = \sum_{i=1}^{n} \log \big(f(y_i;\, u(x_i, \beta)) \big) \tag{2}$$

TDA's `fml` command described in **6.11.2** can be used to implement this general approach.

However, while this approach is very general, it requires a complete specification of the (conditional) density of the given observations. It is often more practical to focus on the conditional expectation of the dependent variable. The modeling approach then becomes

$$\mathrm{E}(Y \mid x) = u(x, \beta) \tag{3}$$

Deriving a likelihood function depends, of course, on assuming some density function for the observations. In general, if the expectation is not directly given by one of the distributional parameters, this requires a more or less complicated re-parameterization.

A relatively flexible approach is possible by using the exponential family of distributions having the density

$$f(y; \theta, \phi) = \exp \left\{ \frac{y\,\theta - b(\theta)}{a(\phi)} + c(y, \phi) \right\} \tag{4}$$

where $a(\phi)$, $b(\theta)$ and $c(y, \phi)$ are some functions of the distributional parameters ϕ and θ. The expectation is

$$\mathrm{E}(Y) = b'(\theta) = \frac{\partial b(\theta)}{\partial \theta} \tag{5}$$

and the variance is given by

$$\mathrm{Var}(Y) = a(\phi)\, b''(\theta) = a(\phi)\, \frac{\partial^2 b(\theta)}{\partial \theta^2} \tag{6}$$

$b''(\phi)$ is often called *variance function* (of the distribution).

This exponential family of distributions has many familiar distributions as special cases and, furthermore, provides a comfortable starting point for the modeling strategy (3). The required re-parameterization is simply

$$\mathrm{E}(Y \mid x) = b'(\theta) \equiv u(x, \beta) \tag{7}$$

An additional simplification is possible if one assumes that the scale parameter, ϕ, does not vary across individual cases. It is possible, then, to estimate β and ϕ separately.

This exponential family of distributions has become the starting point for a broad literature on generalized linear models, see, e.g., McCullagh and Nelder [1983], Aitkin et al. [1989] and, in particular, for the GLIM program, see Francis et al. [1994]. Starting from (7), one needs the specification of $b(\theta)$ and $u(x, \beta)$. Specification of $b(\theta)$ depends on which member of the exponential family of distributions will be used; some options are discussed in later sections. Specification of $u(x, \beta)$ can, in principle, be arbitrary. Generalized linear models as normally defined and implemented, for instance, in the GLIM package (Francis et al., 1994), use a two-part specification. One part is the *linear predictor*

$$\eta = x\beta \tag{8}$$

optionally including an intercept term. The second part is the *link function*

$$\eta = g(\mu), \qquad \mu = \mathrm{E}(Y \mid x) \tag{9}$$

linking the conditional expectation, μ, to the linear predictor, η. The link function must be invertible to allow the calculation of

$$\mu = g^{-1}(\eta) = g^{-1}(x\beta) \tag{10}$$

We will also assume that the link function is twice continuously differentiable. The relationship with the link function u, used in (7), is then

$$u(x, \beta) \equiv g^{-1}(\eta) = g^{-1}(x\beta) \tag{11}$$

Secondly, when formulating generalized linear models, it is normally assumed that the scale parameter, ϕ, in (4) is constant across individual cases. We then have that $a(\phi)$, but not ϕ, may depend on case-specific information. In fact, most often there is a further simplification by assuming that

$$a(\phi) = \phi / w \tag{12}$$

where w is some case-specific prior weight, see Francis et al. [1994, p. 260].

Estimation Approach. Using the specification of generalized linear models described above, maximum likelihood estimation can be achieved with an iteratively reweighted least squares procedure, see McCullagh and Nelder [1983, pp. 31–34], Francis et al. [1994, chap. 11]. The log-likelihood is

$$\ell = \sum_{i=1}^{n} \frac{y_i \theta_i - b(\theta_i)}{\phi / w_i} + c(y_i, \phi) \tag{13}$$

In general, maximization requires an iterative procedure. Given a current parameter vector, β, the Newton-Raphson method would calculate a new estimate

$$\beta^* = \beta - H^{-1} \frac{\partial \ell}{\partial \beta} \tag{14}$$

where H is the Hessian matrix

$$H = \left(\frac{\partial^2 \ell}{\partial \beta_j \partial \beta_k} \right)$$

Assuming m components in the parameter vector, this is an (m, m) matrix. A somewhat simpler approach uses the expectation of H which can be substituted by

$$\tilde{H} = \mathrm{E} \left(\frac{\partial \ell}{\partial \beta_j} \frac{\partial \ell}{\partial \beta_k} \right) = -\mathrm{E}\,(H)$$

Considering first the gradient for the ith component of the log-likelihood, we find

$$\frac{\partial \ell_i}{\partial \beta_j} = \frac{\partial \ell_i}{\partial \theta_i} \frac{\partial \theta_i}{\partial \mu_i} \frac{\partial \mu_i}{\partial \eta_i} \frac{\partial \eta_i}{\partial \beta_j}$$

Using the model specification defined above, this becomes

$$\frac{\partial \ell_i}{\partial \beta_j} = \frac{y_i - \mu_i}{a_i(\phi)} \frac{1}{b''(\theta_i)} \frac{1}{g'(\mu_i)} x_{ij} \tag{15}$$

Using the abbreviations

$$d_i = g'(\mu_i) \quad \text{and} \quad u_i = \frac{w_i}{\phi\, b''(\theta_i)\, d_i^2}$$

this may also be written as

$$\frac{\partial \ell_i}{\partial \beta_j} = (y_i - \mu_i) u_i d_i x_{ij} \tag{16}$$

Since $\partial \ell_i / \partial \eta_j = 0$ if $i \neq j$, we find

$$\tilde{H}_{jk} = \mathrm{E}\left(\frac{\partial \ell}{\partial \beta_j} \frac{\partial \ell}{\partial \beta_k} \right) = \sum_{i=1}^{n} u_i^2 d_i^2 \, \mathrm{E}\left[(y_i - \mu_i)^2 \right] x_{ij} x_{ik}$$

However, as already noted in (6),

$$\mathrm{E}\left[(y_i - \mu_i)^2 \right] = \frac{\phi}{w_i} b''(\theta_i) = \frac{1}{u_i\, d_i^2}$$

and the expectation of the outer product simplifies into

$$\tilde{H}_{jk} = \mathrm{E}\left(\frac{\partial \ell}{\partial \beta_j} \frac{\partial \ell}{\partial \beta_k} \right) = \sum_{i=1}^{n} u_i x_{ij} x_{ik}$$

Written in matrix notation, we finally have

$$\tilde{H} = X'UX\,, \quad U = \mathrm{diag}\,\{u_1, , \ldots, u_n\}$$

(14) can then be rewritten as

$$\beta^* = \beta + (X'UX)^{-1} \frac{\partial \ell}{\partial \beta} = (X'UX)^{-1} \left[X'UX\beta + \frac{\partial \ell}{\partial \beta} \right] \tag{17}$$

This can be further simplified. Using (16) we can write

$$\frac{\partial \ell}{\partial \beta_j} = (X'Uz)_j = \sum_{i=1}^{n} x_{ij} u_i (y_i - \mu_i) d_i$$

where z is an n-vector with components

$$z_i = (y_i - \mu_i)\, d_i$$

Consequently,

$$\frac{\partial \ell}{\partial \beta} = X'Uz$$

and inserting this into (17), we find

$$\beta^* = (X'UX)^{-1} X'U\,[X\beta + z] \tag{18}$$

This is the basic equation for the iteratively re-weighted least squares procedure. The steps are as follows:

1. Given a parameter vector β of starting values and an invertible link function $\eta_i = g(\mu_i)$ we can calculate

$$\eta_i = x_i\beta \quad \text{and} \quad \mu_i = g^{-1}(\eta_i)$$

Since the link function is assumed to be differentiable, we can also calculate $d_i = g'(\mu_i)$ and the components of the z vector, $z_i = (y_i - \mu_i)d_i$. While this can be done quite generally, we furthermore need the weights, u_i, depending on the scaling and variance functions. As noted above, we shall assume that the scaling function is ϕ/w_i, where w_i is some case-specific weight. It is possible, therefore, to ignore the scaling function when estimating the parameter vector β.[1] On the other hand, we need the variance function, $b''(\theta)$, that depends on which member of the exponential family of distributions has been selected for model specification. Several standard options will be described below. Given such a specification, we know the variance function and can calculate the working weights

$$\tilde{u}_i = \frac{w_i}{b''(\theta_i)\, d_i^2}$$

[1] Of course, we need this function for estimating a covariance matrix, see below.

2. The next step is to solve the weighted least squares problem defined in (18) using the working weights $\tilde{U} = \text{diag}\{\tilde{u}_1, \ldots, \tilde{u}_n\}$. This least squares problem may be written as

$$(X\beta + z - X\beta^*)' \tilde{U} (X\beta + z - X\beta^*) \longrightarrow \min$$

with β the given parameter vector for the current iteration and β^* the solution vector. For solving this least squares problem, TDA uses the Hanson and Haskell [1982] algorithm described in Chapter 6.9.1. As an option, this algorithm allows equality and inequality constraints as described below.

3. The procedure is iterated until the difference between β and β^* becomes small, or a maximum number of iterations has been performed. TDA assumes convergence if

$$\max_j \left\{ \frac{|\beta_j^* - \beta_j|}{\max\{\beta_j^*, 1\}} \right\} \leq \texttt{TOLSP} \tag{19}$$

where `TOLSP` is some small value.

4. If the procedure successfully converged, one can estimate a covariance matrix

$$\text{Cov}(\hat{\beta}) = (X'UX)^{-1} = \hat{\phi}\,(X'\tilde{U}X)^{-1} \tag{20}$$

using some estimate of the scale parameter, ϕ. These are the basic steps performed by TDA to estimate generalized linear models. As indicated, the procedure needs a variance function depending on which member of the exponential family of distributions has been selected.

Estimating the Scale Parameter. Not all members of the exponential family of distributions do have a scale parameter. If there is a scale parameter, estimation can be done separately, using an estimate of the structural parameter vector, $\hat{\beta}$. The log-likelihood for estimating ϕ is

$$\ell(\phi) = \sum_{i=1}^{n} w_i \frac{y_i\hat{\theta}_i - b(\hat{\theta}_i)}{\phi} + c(y_i, \phi) \tag{21}$$

Solutions of maximizing this log-likelihood will be called ML-based estimates of the scale factor. Alternatively, one can use the concept of

deviance to estimate a scale parameter, see Francis et al. [1994, p. 270]. This estimate is defined by

$$\hat{\phi}_d = D/d_f \tag{22}$$

where D denotes the deviance (see below) and $d_f = n - \text{rank}(X)$ is the number of degrees of freedom. $\hat{\phi}_d$ will be called deviance-based estimate of the scale parameter.

Goodness-of-Fit Indicators. Two types of goodness-of-fit indicators are often used when estimating generalized linear models. First, a generalized Pearson statistic, defined by

$$X^2 = \sum_{i=1}^{n} \frac{(y_i - \hat{\mu}_i)^2}{V(\hat{\mu}_i)}, \quad \hat{\mu}_i = g^{-1}(x_i\hat{\beta}), \quad V(\hat{\mu}_i) = b''(\hat{\theta}_i) \tag{23}$$

The second indicator is called *deviance* and is derived from considering likelihood ratios, see Francis et al. [1994, p. 274]. If $M_1 \subset M_2$ are two nested models, the likelihood ratio is $2(\ell_2 - \ell_1)$; ℓ_1 and ℓ_2 being the maxima of the corresponding log-likelihood functions. The idea behind the concept of deviance is to compare the log-likelihood of the current model, ℓ_c, with the log-likelihood of a saturated model, ℓ_s, defined as a model having sufficient parameters to reproduce the observed data exactly.

$$D_s = 2\,(\,\ell_s - \ell_c\,)$$

is called the *scaled deviance* (of the current model). Let $\hat{\theta}_{i,c}$ and $\hat{\theta}_{i,s}$ denote estimates for the current and saturated model, respectively, the latter being defined by $b'(\hat{\theta}_{i,s}) = y_i$. Also assume that we use the same estimate, $\hat{\phi}$, for the current and for the saturated model. Then

$$D_s = 2\sum_{i=1}^{n} w_i \frac{y_i(\hat{\theta}_{i,s} - \hat{\theta}_{i,c}) - (b(\hat{\theta}_{i,s}) - b(\hat{\theta}_{i,c}))}{\hat{\phi}} \tag{24}$$

Without scaling,

$$D = \hat{\phi}\, D_s \tag{25}$$

is called simply *deviance*. Detailed information about how TDA calculates both goodness-of-fit indicators will be given below, separately for each type of distribution.

6.15.2 The `glm` Command

The TDA command to estimate generalized linear models is `glm`. The syntax is shown in Box 1. The command allows to select one of five members of the exponential family of distributions, default is `d=1`, selecting a normal distribution. Details will be described in later sections, separately for each type of distribution.

1. A list of variable names must be given with the `v` parameter, the syntax is

```
v = Y,X1,...XM,
```

The right-hand side is a comma-separated list of variable names, optionally containing namelists. The first variable on the right-hand side is used as dependent variable. The remaining variables are used to create the linear predictor. If the dependent variable is followed by m variable names, say $X_1, \ldots, X_m$, the linear predictor is

$$\eta_i = \beta_0 + \sum_{j=1}^{m} X_{ij}\beta_j$$

By default, the linear predictor contains an intercept term, β_0. The `ni=1` parameter can be used to suppress this term. In this case, there must be at least one independent variable.

2. An additional parameter, `yw`, is used to specify a dependent variable as proportions. This will only be recognized if the distribution is binomial, see **6.15.2.2**.

3. The `d` parameter can be used to select a specific type of distribution. Default is `d = 1`, i.e, the normal distribution. For a discussion of specific distributions see subsequent sections.

4. There are two possibilities to specify a link function. One can use one of the pre-defined link functions, or one can explicitly define a link function on the right-hand side of the command. Pre-defined link functions can be selected with the `link` parameter. For available options and defaults, see Box 2. If a link function is specified on the right-hand side of

Box 1 Syntax for `glm` command

```
glm (
      d=...,            type of distribution, def. 1
                        1 = Normal
                        2 = Binomial
                        3 = Poisson
                        4 = Gamma
                        5 = Inverse Gaussian
      link=...,         type of link function (if not user-defined)
                        1 = identity
                        2 = log
                        3 = logit
                        4 = reciprocal
                        5 = probit
                        6 = complementary log-log
                        7 = square root
                        8 = quadratic inverse
      v=...,            list of variables
      yw=...,           additional variable to define proportions
      ni=...,           1 if linear predictor without intercept, def. 0
      lsecon=...,       equality constraints
      lsicon=...,       inequality constraints
      mxit=...,         maximum number of iterations, def. 20
      tolsp=...,        tolerance for parameter change, def. 1.e-8
      xp=...,           starting values
      dsv=...,          input file with starting values
      tfmt=...,         print format for results, def. tfmt=10.4
      ppar=...,         print estimated coefficients to output file
      pcov=...,         print covariance matrix to output file
      mfmt=...,         print format for pcov option, def. 12.4
      pres=...,         print data and estimated values to output file
      fmt=...,          print format for pres option, def. 10.4
      dtda=...,         TDA description of file written with pres
      prot=...,         request protocol file
      pfmt=...,         print format for protocol file, def. 19.11
      ab=...,           domain of user-defined link function
                        def. ab=0,1
) = user-defined link function;
```

Box 2 Built-in link functions

link	function	default if
1	$\eta = \mu$	d = 1
2	$\eta = \log(\mu)$	d = 3
3	$\eta = \log(\mu/(1-\mu))$	d = 2
4	$\eta = 1/\mu$	d = 4
5	$\eta = \Phi^{-1}(\mu)$	
6	$\eta = \log(-\log(1-\mu))$	
7	$\eta = \sqrt{\mu}$	
8	$\eta = 1/\mu^2$	d = 5

the command, the syntax must follow the conventions for the definition of functions in TDA explained in 5.3. The function must contain exactly one parameter and must not refer to data matrix variables. An arbitrary name can be used for the function argument; in the examples below we will use the name `mue`.

The algorithm for estimating generalized linear models requires the link function to be invertible (monotone) and differentiable. If the link function is $\eta = g(\mu)$, the algorithm needs $\mu = g^{-1}(\eta)$ and $g'(\mu)$. Derivatives are calculated by automatic differentiation. For calculating μ, given η, for a user-defined link function, the algorithm also needs to know the domain of the link function, that is, some interval $\mu_a < \mu < \mu_b$. In general, the domain will depend on the selected distribution and link function. Default is $0 < \mu < 1$. To change this default, one can use the parameter

`ab =` μ_a, μ_b`,`

To illustrate user-defined link functions, here are some standard examples:

Identity	`glm(...) = mue;`
Log	`glm(...) = log(mue);`
Logit	`glm(...) = log(mue / (1 - mue));`
Reciprocal	`glm(...) = 1 / mue;`
Probit	`glm(...) = ndi(mue);`
Compl. log-log	`glm(...) = log(-log(1 - mue));`
Square root	`glm(...) = sqrt(mue);`

5. As explained in 6.15.1, estimation is done with an iteratively reweighted least squares procedure. The behavior of the algorithm can be controlled with two parameters. The `mxit` parameter can be used to define the maximum number of iterations, default is `mxit=20`. The `tolsp` parameter can be used to define the convergence tolerance according to (19), default is `tolsp = 1.e-8`. While the command performs the iterations, a message about the current value of the scaled parameter change is written into the standard error output. To get additional information about the iterations, one can request a protocol file with the parameter

```
prot = name_of_an_output_file,
```

The print format can be controlled with the `pfmt` parameter.

6. There are two options to define starting values for the model parameters, β. First, one can use

```
xp = b1,b2,...,
```

to provide starting values directly in the command file. Alternatively, one can use

```
dsv = fname,
```

where `fname` is the name of a free-format input file containing starting values in the first numerical column. If the model has m parameters, up to m values are taken from the right-hand side of the `xp` parameter, or read from the input file defined with the `dsv` parameter. By default, all parameters get the starting value zero.

7. Basic estimation results are written into the standard output. Examples will be given below. The print format can be controlled with the `tfmt` parameter, default is `tfmt=10.4`.

8. As an option, estimated parameters and standard errors can be written into an output file using the parameter

```
ppar = name_of_an_output_file,
```

The print format is the same as used for the standard output and may be changed with the `tfmt` parameter.

9. Also optionally, one can write the estimated covariance matrix into an output file using the parameter

```
pcov = name_of_an_output_file,
```

The print format can be controlled with the `mfmt` parameter, default is `mfmt=12.4`.

10. As a further option, one can use the parameter

```
pres = name_of_an_output_file,
```

to request an output file containing the following entries:

Case	Y	$\hat{Y}$	η	X_1	...	X_m	W	$\tilde{U}$
1	y_1	$\hat{y}_1$	η_1	x_{11}	$\cdots$	x_{1m}	w_1	$\tilde{u}_1$
2	y_2	$\hat{y}_2$	η_2	x_{21}	$\cdots$	x_{2m}	w_2	$\tilde{u}_2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$

11. Finally, one can define equality and inequality constraints with the `lsecon` and `lsicon` parameters, respectively. For an explanation of the syntax, see **6.9.1**. Note that a covariance matrix for the estimated parameters is only calculated if there are no inequality constraints.

6.15.2.1 Normal Distribution

As a first member of the exponential family of distributions, one gets the normal distribution by specifying

$$a(\phi) = \phi, \quad b(\theta) = \frac{1}{2}\theta^2, \quad c(y,\phi) = -\frac{1}{2}\left(\frac{y^2}{\phi} + \log(2\pi\phi)\right)$$

Inserting these expressions into (4) and using

$$\mu \equiv \theta, \quad \sigma^2 \equiv \phi$$

one gets the normal density function

$$f(y;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2\right\}$$

The variance function is simply $b''(\theta) = 1$. Given an invertible link function $\eta_i = g(\mu_i)$, the relation between the mean value of the distribution and the linear predictor is

$$\mu_i = g^{-1}(x_i\beta)$$

If $g(\mu_i)$ is the identity function, the resulting model is an ordinary linear regression model. Of course, many other link functions can be used. Note that, in general, the default domain for the link function will not be appropriate for the normal distribution and should be redefined with the `ab` parameter when user defined link functions are used.

The normal distribution has a scale parameter equal to its variance. The log-likelihood for estimating this scale parameter is

$$\ell(\phi) = \sum_{i=1}^{n} w_i \left[\frac{y_i\hat{\mu}_i - \hat{\mu}_i^2/2}{\phi} - \frac{1}{2}\left(\frac{y_i^2}{\phi} + \log(2\pi\phi)\right)\right]$$

Maximizing this log-likelihood gives the estimate

$$\hat{\phi} = \frac{\sum_{i=1}^{n} w_i(y_i - \hat{\mu}_i)^2}{\sum_{i=1}^{n} w_i}$$

Box 1 Command file `glm1.cf`

```
nvar(
    dfile = lsreg1.dat,     # data file
    Height = c1,
    Weight = c2,
);
glm(    # use default normal distribution, identity link

    v = Weight,Height,      # variables

    pcov = cov,             # write covariance matrix to cov
    ppar = par,             # write parameter to par
    pres = res,             # write residuals to res
);
```

TDA uses this formula to calculate an ML estimate for the scale parameter. Note that this is different from the deviance-based estimate of the scale parameter, see below.

For the normal distribution, the variance function is $b''(\theta) = 1$ and the generalized Pearson statistic is

$$X^2 = \sum_{i=1}^{n} (y_i - \hat{\mu}_i)^2$$

The deviance is calculated according to (24) and (25). The saturated model is defined by $\hat{\theta}_{i,s} = y_i$ and $b(\hat{\theta}_{i,s}) = y_i^2/2$. Inserting this into (24), one gets

$$D = \sum_{i=1}^{n} w_i (y_i - \hat{\mu}_i)^2$$

TDA uses this formula to calculate the deviance, and the deviance-based estimate of the scale factor, $\hat{\phi}_d = D/d_f$.

Example 1 To illustrate the `glm` command, we begin with a replication of the simple linear regression model discussed in the first example in 6.9.1. The command file, `glm1.cf`, is shown in Box 1, part of the standard output is shown in Box 2. The command file uses a normal distribution and an identity link function (both being defaults). The same results can be achieved with a user-defined link function, for example, by using the command

```
glm (..., ab=1.e-6,1.e6) = mue;
```

Box 2 Part of standard output from `glm1.cf`

```
Distribution: normal.
Link function: identity.

Variables (cross-section)
-------------------------
Y   : Weight
X1  : Height

Estimation with iteratively re-weighted least squares.
Number of model parameters: 2
Maximum number of iterations: 20
Tolerance for scaled parameter change: 1e-06

Data and estimated values written to: res

Convergence reached in 2 iterations.
Final scaled parameter change: 0

Rank of data matrix: 2
Degrees of freedom: 17
Deviance                          2142.4880
Pearson statistic                 2142.4880
ML-based scaling factor            112.7625
Deviance-based scaling factor      126.0287

Idx  Wave  Variable       Coeff      Error    Coeff/E  Signif
----------------------------------------------------------------
  1     -  Intercept  -143.0269    32.2746    -4.4316  0.9996
  2     1  Height        3.8990     0.5161     7.5549  1.0000

Parameter estimates written to: par
Covariance matrix written to: cov
```

Box 3 Command file `glm2.cf`

```
nvar(
    noc = 100,
    X = rd(-3,3),
    Y = exp(3 + X) + rd,
);
glm(
    link = 2,           # logarithmic link function
    v = Y,X,            # variables
    xp = 5,5,           # some bad starting values
);
```

Box 4 Part of standard output from `glm2.cf`

```
Distribution: normal.
Link function: log.

Variables (cross-section)
-------------------------
Y   : Y
X1  : X

Estimation with iteratively re-weighted least squares.
Number of model parameters: 2
Maximum number of iterations: 20
Tolerance for scaled parameter change: 1e-06
Got 2 starting value(s) from xp parameter.

Convergence reached in 19 iterations.
Final scaled parameter change: 4.13314e-10

Rank of data matrix: 2
Degrees of freedom: 98
Deviance                              19.9937
Pearson statistic                     19.9937
ML-based scaling factor                0.1999
Deviance-based scaling factor          0.2040

Idx  Wave  Variable       Coeff      Error    Coeff/E  Signif
-----------------------------------------------------------------
  1     -  Intercept     3.0128     0.0020  1524.0440  1.0000
  2     1  X             0.9958     0.0008  1299.1800  1.0000
```

Of course, we need to re-define the domain for the link function with the `ab` parameter. The modified command file is supplied as `glm1a.cf` in the example archive.

Example 2 To illustrate a logarithmic link function we use some random data, see command file `glm2.cf` in Box 3. Data generation is by

$$y_i = \exp(3 + x_i) + r_i$$

where r_i and x_i are random numbers, equally distributed in $(0, 1)$ and in $(-3, 3)$, respectively. Results are shown in Box 4. The parameters are quite well recovered. (The example archive contains a command file, `glm2a.cf`, that estimates the same model with a user-defined link function.)

6.15.2.2 Binomial Distribution

If the dependent variable, Y, has possible values in $\{0, 1, \ldots, m\}$, where $m \geq 1$, we may assume a binomial distribution with density function

$$f(y) = \binom{m}{y} \pi^y (1-\pi)^{m-y} \qquad 0 \leq \pi \leq 1 \tag{1}$$

This is again a special case of the exponential family of distributions. This is easily seen by using the definitions

$$\begin{aligned} \theta &= \log\left(\frac{\pi}{1-\pi}\right), \ \pi = \frac{\exp(\theta)}{1+\exp(\theta)} \\ \phi &= 1 \\ b(\theta) &= m\log(1+\exp(\theta)) \\ c(y,\phi) &= \log\left\{\binom{m}{y}\right\} \end{aligned}$$

Inserting these expressions into 6.15.1-4 will give (1). The expectation is

$$\mathrm{E}(Y) = b'(\theta) = m\pi$$

and the variance is

$$\mathrm{Var}(Y) = m\pi(1-\pi)$$

This shows that models for a dependent variable with a binomial distribution can be specified as special cases of generalized linear models. In the `glm` command, the `d=2` parameter selects the binomial distribution.

Proportions. We speak of individual-level data if $m = 1$. The dependent variable is then expected to take values in $\{0, 1\}$. Note that, contrary to TDA's `qreg` command, the `glm` command expects that the variable Y actually has values in $\{0, 1\}$, there is no internal recoding.

On the other hand, if $m > 1$, we speak of grouped data, or *proportions*. In order to tell the `glm` command that it should use proportions,

instead of individual-level data, one needs to specify an additional variable, say M, that contains the values of m. This is done with the additional parameter

```
yw = M,
```

M must be the name of a variable that is defined in the currently active data matrix and must contain only positive integer values. And, of course, if Y is the name of the dependent variable, we need to have

$$0 \leq \mathtt{Y}_i \leq \mathtt{M}_i \quad \text{and} \quad 1 \leq \mathtt{M}_i$$

Deviance. Since $\phi = 1$, deviance and scaled deviance are identical. Calculation follows formula 6.15.1-24. For the saturated model, we assume that

$$y_i = \hat{\mu}_{i,s} = b'(\hat{\theta}_{i,s}) = m_i \hat{\pi}_{i,s}$$

and, consequently,

$$\hat{\theta}_{i,s} = \log\left(\frac{\hat{\pi}_{i,s}}{1-\hat{\pi}_{i,s}}\right) = \log\left(\frac{y_i}{m_i - y_i}\right)$$

For the current model we find analoguously

$$\hat{\theta}_{i,c} = \log\left(\frac{\hat{\pi}_{i,c}}{1-\hat{\pi}_{i,c}}\right) = \log\left(\frac{\hat{\mu}_i}{m_i - \hat{\mu}_i}\right)$$

By the same reasoning, we also find

$$b(\hat{\theta}_{i,s}) = m_i \log\left(\frac{1}{1 - y_i/m_i}\right)$$

$$b(\hat{\theta}_{i,c}) = m_i \log\left(\frac{1}{1 - \hat{\mu}_i/m_i}\right)$$

It might happen that the expressions for the saturated model cannot be evaluated. However, inserting into 6.15.1-24 gives

$$D_s = 2\sum_{i=1}^{n} w_i \left\{ y_i \log\left(\frac{y_i}{\hat{\mu}_i}\right) + (m_i - y_i)\log\left(\frac{m_i - y_i}{m_i - \hat{\mu}_i}\right)\right\}$$

TDA uses this formula for calculating the deviance in case of a binomial distribution.

Box 1 Command file `glm3.cf`

```
nvar(
    dfile = qr1.dat,
    Dose = c1,
    Weight = c2,
    Response = c3,
    Log10Dose = log(Dose) / log(10),
);
cwt = Weight;   use case weights

glm(
    d = 2,                    # binomial distribution
    v = Response,Log10Dose, # variables
);
```

Logit Link Function. We arrive at a standard logit model if we assume that

$$\frac{\mu_i}{m_i} = \frac{\exp(\eta_i)}{1+\exp(\eta_i)}$$

with the linear predictor $\eta_i = x_i\beta$. This corresponds to a logit link function that is the default for the binomial distribution. In order that individual-level and grouped data can be treated in the same way, all link functions for the binomial case will be interpreted as

$$g(\mu_i/m_i) = \eta_i$$

If g is then specified as a logit function we have

$$\eta_i = \log\left(\frac{\mu_i/m_i}{1-\mu_i/m_i}\right)$$

Example 1 To illustrate we replicate example **6.12.2**-1. Box 1 shows the command file, Box 2 shows part of the standard output. The case weights defined with the `cwt` command are used by the `glm` command. Estimation results are identical with the results from the `qreg` command. The only difference is in the column `Signif` due to the fact that the `qreg` command ues a normal distribution and the `glm` command, instead, a t-distribution.

Example 2 To illustrate estimation of logit model with grouped data we use the data file `glm3.dat` shown in Box 3. The data are taken from

Box 2 Part of standard output from `glm3.cf`

```
Distribution: binomial.
Link function: logit.

Variables (cross-section)
-------------------------
Y   : Response
X1  : Log10Dose

Estimation with iteratively re-weighted least squares.
Using weights defined by: Weight
Number of model parameters: 2
Maximum number of iterations: 20
Tolerance for scaled parameter change: 1e-06

Convergence reached in 6 iterations.
Final scaled parameter change: 2.75435e-16

Rank of data matrix: 2
Degrees of freedom: 11
Deviance                          74.2213
Fixed scaling factor               1.0000

Idx  Wave  Variable        Coeff      Error    Coeff/E  Signif
-------------------------------------------------------------
  1     -  Intercept     -3.2246     0.8861    -3.6393  0.9961
  2     1  Log10Dose      5.9702     1.4492     4.1197  0.9983
```

Box 3 Data file `glm3.dat`

```
  D     N   Y
-------------
 0.01  49   0
 2.60  50   6
 3.80  48  16
 5.10  46  24
 7.70  49  42
10.20  50  44
```

Francis et al. [1994, p. 440]. The command file, `glm4.cf` (Box 4) contains two `glm` commands. Both estimate the same logit model. The first command uses the build-in logit link function (default with the binomial distribution), the second one uses a user-defined link function. Estimation results from the first command are shown in Box 5. Of course, the

Box 4 Command file `glm4.cf`

```
nvar(
    dfile = glm3.dat,
    D = log(c1),
    N = c2,
    Y = c3,
);
glm(
    d = 2,          # binomial distribution
    v = Y,D,        # variables
    yw = N,         # count
);
glm(
    d = 2,          # binomial distribution
    v = Y,D,        # variables
    yw = N,         # count
) = log(mue / (1 - mue));   # user-defined link function
```

second `glm` command should give the same results.

Probit Link Function. Substituting the logit link function by a probit function, i.e.,

$$\eta_i = \Phi^{-1}(\mu_i/m_i)$$

results in a probit model. Box 6 shows the command file `glm4a.cf` that replicates example 2 with a probit model. The first `glm` command uses the `link=5` parameter to select the in-build probit function, the second `glm` command uses an equivalent user-defined link function. (The `ndi` operator evaluates the inverse of the standard normal distribution function.)

Box 5 Part of standard output from `glm4.cf`

```
Distribution: binomial.
Link function: logit.

Variables (cross-section)
-------------------------
Y   : Y
X1  : D

Variable used to define proportions: N

Estimation with iteratively re-weighted least squares.
Number of model parameters: 2
Maximum number of iterations: 20
Tolerance for scaled parameter change: 1e-06

Convergence reached in 8 iterations.
Final scaled parameter change: 2.86182e-16

Rank of data matrix: 2
Degrees of freedom: 4
Deviance                          1.4241
Fixed scaling factor              1.0000

Idx  Wave  Variable       Coeff      Error    Coeff/E  Signif
----------------------------------------------------------------
  1     -  Intercept    -4.8869     0.6429    -7.6010  0.9984
  2     1  D             3.1035     0.3877     8.0047  0.9987
```

Box 6 Command file `glm4a.cf`

```
nvar(...);          # same as in glm4.cf
glm(
    d = 2,          # binomial distribution
    link = 5,       # probit link function
    v = Y,D,        # variables
    yw = N,         # count
);
glm(
    d = 2,          # binomial distribution
    v = Y,D,        # variables
    yw = N,         # count
) = ndi(mue);       # user-defined link function
```

6.15.2.3 Poisson Distribution

The Poisson distribution has density

$$f(y) = e^{-\lambda} \frac{\lambda^y}{y!} \qquad \lambda > 0 \tag{1}$$

where y takes values in $\{0, 1, 2, \ldots\}$. It is therefore often used to model count data. The Poisson distribution is a special case of the exponential family with

$$\begin{aligned} \theta &= \log(\lambda)\,, \lambda = e^\theta \\ \phi &= 1 \\ b(\theta) &= e^\theta \\ c(y, \phi) &= -\log(y!) \end{aligned}$$

The expectation is

$$\mathrm{E}(Y) = b'(\theta) = \lambda = e^\theta$$

and the variance is

$$\mathrm{Var}(Y) = \lambda = e^\theta$$

Thus the variance is equal to the expectation. The Poisson distribution can be specified with the `d=3` parameter in the `glm` command. The default link function is the logarithm (`link=2`). The `glm` command expects the variable Y to have values in $\{0, 1, 2, \ldots\}$.

Deviance. Since $\phi = 1$, deviance and scaled deviance are identical. Calculation follows formula (24). For the saturated model, we assume that

$$y_i = \hat{\mu}_{i,s} = \hat{\lambda}_{i,s}$$

Thus

$$\hat{\theta}_{i,s} = \log\left(\hat{\lambda}_{i,s}\right)$$

Box 1 Data file `glm5.dat`

```
     X     Y
 ---------
 0.844   7
 0.603   5
 0.316   6
 0.245   2
 0.944   6
-0.322   1
 0.365   3
-0.386   3
-0.795   1
-0.609   0
```

Box 2 Command file `glm5.cf`

```
nvar(
     dfile = glm5.dat,
     X     = c1,   # covariate
     Y     = c2,
  );

glm(
    d = 3,       # Poisson distribution
    v = Y,X,     # variables
  );
```

For the current model we find

$$\hat{\theta}_{i,c} = \log\left(\hat{\lambda}_{i,c}\right)$$

Consequently,

$$b(\hat{\theta}_{i,s}) = \hat{\lambda}_{i,s} = y_i$$

$$b(\hat{\theta}_{i,c}) = \hat{\lambda}_{i,c}$$

Inserting these expressions into 6.15.1-24 gives

$$D = 2\sum_{i=1}^{n} w_i \left\{ y_i \log\left(\frac{y_i}{\hat{\lambda}_{i,c}}\right) - (y_i - \hat{\lambda}_{i,c}) \right\}$$

where we set $0\log(0) = 0$.

Box 3 Part of standard output from `glm5.cf`

```
Distribution: Poisson.
Link function: log.

Variables (cross-section)
-------------------------
Y   : Y
X1  : X

Estimation with iteratively re-weighted least squares.
Number of model parameters: 2
Maximum number of iterations: 20
Tolerance for scaled parameter change: 1e-06

Convergence reached in 9 iterations.
Final scaled parameter change: 2.22045e-16

Rank of data matrix: 2
Degrees of freedom: 8
Deviance                              6.1902

Idx  Wave  Variable       Coeff       Error    Coeff/E  Signif
-----------------------------------------------------------------
  1     -  Intercept     0.8673      0.2384     3.6382  0.9934
  2     1  X             1.1754      0.3565     3.2974  0.9891
```

Example 1 To illustrate we use the data in Box 1 with the command file `glm5.cf` (Box 2). Part of the output is shown in Box 3.

Box 4 Command file `glm6.cf`

```
nvar(
     dfile = cd1.dat,
     isel = ge(c1,0),    # select valid cases

     NDI    = c1,
     Service= c2,
     B      = c3,
     C      = c4,
     D      = c5,
     E      = c6,
     C60    = c7,
     C65    = c8,
     C70    = c9,
     P75    = c10,
     LOGS   = log(Service),
    );

glm( d = 3,
     v = NDI, B, C, D, E, C60, C65, C70, P75, LOGS,
     lsecon = b9 = 1,   # constraint: blogs = 1
     mxit = 100,
   );
```

Example 2 In this example we use the ship damage data from McCullagh and Nelder [1983, p. 136–140]. The command file is `glm6.cf` (Box 4). It demonstrates the use of equality constraints: The effect of the aggregate months of service should be proportional to the expected number of damage incidents. The coefficient of the logarithm of service time is thus constraint to be 1. Also the number of iterations must be increased to achieve convergence. Part of the output is shown in (Box 5). It replicates the results in McCullagh and Nelder [1983, p. 139] except for the standard errors. McCullagh and Nelder use a factor of approximately 1.3 from an overdispersion model.

Box 5 Part of standard output from `glm6.cf`

```
Distribution: Poisson.
Link function: log.

Variables (cross-section)
-------------------------
Y   : NDI
X1  : B
X2  : C
X3  : D
X4  : E
X5  : C60
X6  : C65
X7  : C70
X8  : P75
X9  : LOGS

Estimation with iteratively re-weighted least squares.
Number of model parameters: 10

LSECon: 1 b9 = 1
Equality constraints: 1
Inequality constraints: 0

Maximum number of iterations: 100
Tolerance for scaled parameter change: 1e-06

Convergence reached in 41 iterations.
Final scaled parameter change: 1.66533e-15

Rank of data matrix: 9
Degrees of freedom: 25
Deviance                          38.6950

Idx  Wave  Variable       Coeff      Error    Coeff/E  Signif
-----------------------------------------------------------------
  1     -  Intercept    -6.4059     0.2174   -29.4600  1.0000
  2     1  B            -0.5433     0.1776    -3.0595  0.9948
  3     1  C            -0.6874     0.3290    -2.0891  0.9530
  4     1  D            -0.0760     0.2906    -0.2614  0.2041
  5     1  E             0.3256     0.2359     1.3803  0.8203
  6     1  C60           0.6971     0.1496     4.6587  0.9999
  7     1  C65           0.8184     0.1698     4.8207  0.9999
  8     1  C70           0.4534     0.2332     1.9446  0.9368
  9     1  P75           0.3845     0.1183     3.2507  0.9967
 10     1  LOGS          1.0000        ---        ---     ---
```

6.15.2.4 Gamma Distribution

The gamma distribution has density

$$f(y) = \frac{1}{\Gamma(\nu)} \left(\frac{\nu}{\mu}\right)^{\nu} y^{\nu-1} \exp\left(-\frac{\nu y}{\mu}\right) \qquad \nu, \mu > 0 \tag{1}$$

where y takes values in the non-negative reals $\mathbb{R}_+$. The gamma distribution is a special case of the exponential family with

$$\begin{aligned} \theta &= -\frac{1}{\mu}, \mu = -\frac{1}{\theta} \\ \phi &= \frac{1}{\nu} \\ b(\theta) &= -\log(-\theta) \\ c(y,\phi) &= (\log(y) - \log(\phi))/\phi - \log(\Gamma(1/\phi)) \end{aligned}$$

The expectation is

$$\mathrm{E}(Y) = b'(\theta) = -\frac{1}{\theta} = \mu$$

and the variance is

$$\mathrm{Var}(Y) = b''(\theta)\phi = \frac{\mu^2}{\nu}$$

Thus the variance is proportional to the square of the expectation. In other words, the coefficient of variation is constant. The gamma distribution can be specified with the `d=4` parameter in the `glm` command. The default link function is the inverse link (`link=4`). This is also the canonical link. But note that this link will not necessarily respect the condition $\mu > 0$. Thus, very often also the log link or the identity link are considered.

Deviance. For the saturated model, we assume that

$$y_i = \hat{\mu}_{i,s}$$

Box 1 Command file `glm7.cf`

```
nvar(
  X = rd,
  Y = -exp(1+X)*(log(rd) + log(rd))/2, # Gamma with form parameter 2
                                       # and expectation exp(1+X)
);
glm(
  d = 4,
  link = 2,
  v = Y,X,
);
clear;

nvar(
  X = rd,
  Y = -(log(rd) + log(rd))/((1+X)*2), # Gamma with form parameter 2
                                      # and expectation 1/(1+X)
);
glm(
  d = 4,
  xp = 1,1,   # starting values required
  mxit = 50,
  v = Y,X,
);
```

If ν is taken to be a constant, the log likelihood reduces to

$$\nu \sum_i w_i \left(-\frac{y_i}{\mu_i} - \log(\mu_i) \right)$$

Since twice the log likelihood of the saturated model takes the value $-2\sum w_i(1 + \log(y_i))$, the deviance can be written as

$$D = -2 \sum_{i=1}^{n} w_i \left\{ \log\left(\frac{y_i}{\hat{\mu}_i}\right) - \frac{y_i - \hat{\mu}_i}{\hat{\mu}_i} \right\}$$

Note that this is only defined for $y_i > 0$.

Estimating ν. Not implemented.

Example 1 We use simulated data as a first illustration. Box 1 shows the command file. It generates gamma distributed variables with covariate effects first with the log link and then with the canonical inverse link. Results are shown in Box 2 and 3.

Box 2 Part of standard output from `glm7.cf`

```
Distribution: gamma.
Link function: log.

Estimation with iteratively re-weighted least squares.
Number of model parameters: 2
Maximum number of iterations: 20
Tolerance for scaled parameter change: 1e-06

Convergence reached in 10 iterations.
Final scaled parameter change: 1.72821e-08

Rank of data matrix: 2
Degrees of freedom: 998
Deviance                          519.1471

Idx  Wave  Variable       Coeff      Error    Coeff/E  Signif
-----------------------------------------------------------------
  1     -  Intercept     0.9620     0.0624    15.4239  1.0000
  2     1  X             1.0947     0.1070    10.2332  1.0000
```

Box 3 Part of standard output from `glm7.cf`

```
Distribution: gamma.
Link function: reciprocal.

Estimation with iteratively re-weighted least squares.
Number of model parameters: 2
Maximum number of iterations: 50
Tolerance for scaled parameter change: 1e-06
2 starting value(s) from xp parameter.

Convergence reached in 3 iterations.
Final scaled parameter change: 4.12683e-15

Rank of data matrix: 2
Degrees of freedom: 998
Deviance                          550.6416

Idx  Wave  Variable       Coeff      Error    Coeff/E  Signif
-----------------------------------------------------------------
  1     -  Intercept     1.0082     0.0748    13.4819  1.0000
  2     1  X             1.0223     0.1604     6.3736  1.0000
```

Box 4 Command file `glm8.cf`

```
nvar(
    dfile = glm8a.dat,
    U = c1,
    Y = c2,
    L = c3,
    X = log(U),
    LX = L * X,
);

tsel=eq(L,0);

glm(
    d = 4,
    v = Y,X,
    xp = -0.02,0.02,    # starting values required
);

tsel=eq(L,1);

glm(
    d = 4,
    v = Y,X,
    xp = -0.02,0.02,    # starting values required
);
```

Example 2 In this example we use the blood clutting data from McCullagh and Nelder [1983, p. 163]. The command file is `glm8.cf` (Box 4). It uses the canonical (inverse) link separately for the two lots. Part of the output is shown in Box 5. It replicates the results in McCullagh and Nelder [1983, p. 163], except for the estimated standard errors. The latter are computed using a moment estimate of the dispersion parameter by McCullagh and Nelder.

Box 5 Part of standard output from `glm8.cf`

```
Distribution: gamma.
Link function: reciprocal.

Estimation with iteratively re-weighted least squares.
Number of model parameters: 2
Maximum number of iterations: 20
Tolerance for scaled parameter change: 1e-06
2 starting value(s) from xp parameter.


Convergence reached in 5 iterations.
Final scaled parameter change: 1.48388e-12

Rank of data matrix: 2
Degrees of freedom: 7
Deviance                              0.0167

Idx  Wave  Variable       Coeff      Error    Coeff/E  Signif
-----------------------------------------------------------------
  1     -  Intercept    -0.0166     0.0188    -0.8827  0.5933
  2     1  X             0.0153     0.0084     1.8287  0.8898

tsel=eq(L,1)

Distribution: gamma.
Link function: reciprocal.
Convergence reached in 4 iterations.
Final scaled parameter change: 2.45984e-15

Rank of data matrix: 2
Degrees of freedom: 7
Deviance                              0.0127

Idx  Wave  Variable       Coeff      Error    Coeff/E  Signif
-----------------------------------------------------------------
  1     -  Intercept    -0.0239     0.0311    -0.7675  0.5321
  2     1  X             0.0236     0.0135     1.7423  0.8750
```

6.15.2.5 Inverse Gaussian Distribution

The inverse Gaussian distribution has density

$$f(y) = \frac{1}{\sqrt{2\pi\sigma^2 y^3}} \exp\left\{-\frac{(y-\mu)^2}{2\sigma^2\mu^2 y}\right\} \qquad \mu, \sigma^2 > 0 \tag{1}$$

where y takes values in the positive reals. The inverse Gaussian distribution is a special case of the exponential family with

$$\begin{aligned} \theta &= -\frac{1}{2\mu^2}, \mu = \frac{1}{\sqrt{-2\theta}} \\ \phi &= \sigma^2 \\ b(\theta) &= -\sqrt{-2\theta} \\ c(y,\phi) &= -\frac{1}{2}\left(\log(2\pi\phi y^3) + \frac{1}{\phi y}\right) \end{aligned}$$

The expectation is

$$\mathrm{E}(Y) = b'(\theta) = -\frac{1}{\sqrt{-2\theta}} = \mu$$

and the variance is

$$\mathrm{Var}(Y) = b''(\theta)\phi = \mu^3\sigma^2$$

Thus the variance is proportional to the cube of the expectation. The inverse Gaussian distribution can be specified with the `d=5` parameter in the `glm` command. The default link function is the quadratic inverse link (`link=5`). This is also the canonical link.

Deviance. For the saturated model, we assume that

$$y_i = \hat{\mu}_{i,s}$$

If σ^2 is taken to be a constant, the log likelihood reduces to

$$\frac{1}{\sigma^2}\sum_i w_i\left(-\frac{y_i}{2\mu_i^2} + \frac{1}{\mu}\right)$$

Since twice the log likelihood of the saturated model takes the value $-\sum w_i/(y_i\sigma^2)$, the deviance can be written as

$$D = \sum_{i=1}^{n} w_i \left\{ \frac{(y_i - \hat{\mu})^2}{y_i \hat{\mu}^2} \right\}$$

Box 1 Command file `glm9.cf`

```
nvar(
  X = rd,
  Mu = 1 + X,
  Z = rdn^2,
  X1 = Mu + Mu^2 * Z - Mu * sqrt(2 * Mu * Z + Mu^2 * Z^2),
  Y = if le(rd,Mu / (Mu + X1)) then X1 else Mu^2 / X1,
);
glm(
  d = 5,
  link = 1,
  xp = 0.5,0.5,
  v = Y,X,
);
```

Box 2 Part of standard output from `glm9.cf`

```
Distribution: inverse Gaussian.
Link function: identity.

Estimation with iteratively re-weighted least squares.
Number of model parameters: 2
Maximum number of iterations: 20
Tolerance for scaled parameter change: 1e-06
2 starting value(s) from xp parameter.

Convergence reached in 6 iterations.
Final scaled parameter change: 7.00738e-09

Rank of data matrix: 2
Degrees of freedom: 998
Deviance                          1933.9975

Idx  Wave  Variable       Coeff      Error    Coeff/E  Signif
----------------------------------------------------------------
  1     -  Intercept     1.0702     0.0867    12.3497  1.0000
  2     1  X             0.8148     0.1909     4.2690  1.0000
```

Example 1 We use simulated data as an illustration. Box 1 shows the command file. It generates inverse Gaussian variables with covariate effects with the identity link using a simulation method described in Seshadri [1993, p. 203–204]. In this case, non-default starting values must be given. Results are shown in Box 2.

6.16 Nonlinear Regression

This chapter deals with nonlinear regression. It includes linear regression as a special case. Currently, there is only a single section.

6.16.1 The `freg` Command

6.16.1 The `freg` Command

If Y_i is a dependent variable for cases $i = 1, \ldots, n$ and X_i is a corresponding vector of covariates, a nonlinear regression model can be written as

$$f(\theta) = \sum_{i=1}^{n} (Y_i - g(X_i, \theta))^2 \quad \longrightarrow \quad \min$$

While it would be possible to use the `fmin` command to find a solution, there are somewhat different conventions for calculating a covariance matrix for parameter estimates. TDA therefore provides a complementary command, `freg`, for nonlinear regression. The syntax and parameters for this command are identical with the `fmin` command. The only difference is that, when using the `freg` command, the covariance matrix is calculated as

$$2 \frac{f(\hat{\theta})}{n - p} H(\hat{\theta})^{-1}$$

n is the number of data matrix cases, p the number of model parameters, $H(\theta)$ is the Hessian of $f(\theta)$, and $\hat{\theta}$ denotes the parameter estimates. This convention makes linear regression a special case of nonlinear regression.

Box 1 Command file `freg1.cf`

```
nvar(
    dfile = freg1.dat,  # data file (Gallant, p. 4)
    Y = c2,
    X1 = c3,
    X2 = c4,
    X3 = c5,
);
freg (
    xp = -0.1,1,-1,-1,      # starting values

) = ax = a1 * X1 + a2 * X2 + a4 * exp(a3 * X3),
    fn = (Y - ax)^2;
```

Example 1 To illustrate the `freg` command we use an example from Gallant [1987, p. 4]. The command file, `freg1.cf`, is shown in Box 1. It creates four variables, `Y, X1, X2, X3`, based on the data file `freg1.dat` containing Gallant's example data. The model is

$$Y = X_1\, a_1 + X_2\, a_2 + a_4\, \exp(X_3\, a_3) + \epsilon$$

The `freg` command specifies the corresponding function to be minimized. Its standard output is shown in Box 2.

For most applications, the `freg` command, like the `fmin` and `fml` commands, requires reasonable starting values.

Box 2 Output from command file `freg1.cf`

```
Nonlinear regression. Current memory: 148695 bytes.

Function definition:
ax      = a1*X1+a2*X2+a4*exp(a3*X3)
fn      = (Y-ax)^2

Function will be summed over 30 data matrix cases.

Nonlinear regression.
Algorithm 5: Newton (I)

Number of model parameters: 4
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: 1

Got starting values from xp parameter.
Idx  Parameter  Starting value
  1  a1              -0.1000
  2  a2               1.0000
  3  a3              -1.0000
  4  a4              -1.0000

Convergence reached in 9 iterations.
Number of function evaluations: 11 (11,11)

Minimum of function: 0.0304955
Norm of final gradient vector: 3.95162e-11
Last absolute change of function value: 4.75342e-11
Last relative change in parameters: 4.7722e-06

Idx  Parameter       Value      Error    Value/E  Signif
-------------------------------------------------------
  1  a1            -0.0259     0.0126    -2.0511  0.9597
  2  a2             1.0157     0.0099   102.4481  1.0000
  3  a3            -1.1157     0.1607    -6.9423  1.0000
  4  a4            -0.5049     0.0255   -19.7862  1.0000

Function (starting values):     1.3728
Function (final estimates):     0.0305
```

6.17 Transition Rate Models

This chapter deals with transition rate models. They require an episode data structure as explained in 3.3. The sections are:

6.17.1 Introduction

This section provides introductory remarks about specification and estimation of transition rate models. Specific types of models are dealt with in subsequent sections. Here we shortly discuss TDA's general approach to the formulation of parametric transition rate models.

6.17.1.1 Parametric Rate Models

For the discussion in this section we assume a set of single episodes:

$$(o_i, d_i, s_i, t_i, x_i) \qquad i = 1, \ldots, N \tag{1}$$

where s_i is the starting time of the episode, t_i is the ending time, o_i is the origin state, d_i is the destination state, and x_i is a (row) vector of covariates which, for the moment, are assumed to have fixed values during the episode to which they are associated. Some of the episode may be right censored, but we assume that they are not left censored.

Formulation (1) allows for different origin and destination states and, therefore, covers the case of multi-episode data. Model estimation generally assumes that the individual episodes are statistically independent and this might not be true for multi-episode data where each individual can contribute more than one episode. But this requirement is conditional on covariates, and can often be met by using covariates to capture information about the past history of the individual episodes.

We use $\mathcal{O}$ and $\mathcal{D}_j$ $(j \in \mathcal{O})$ to refer to the origin and destination state spaces, respectively. Any pair (j, k), $j \in \mathcal{O}$ and $k \in \mathcal{D}_j$, is called a transition from origin state j to destination state k.

The modeling approach will be based on a process time axis, that is, we assume that all episodes begin at time zero ($s_i = 0$ for all $i = 1, \ldots, N$). It is then possible to describe the episodes in terms of durations as discussed in **3.3.1**.

The general setup of the models will be explained in two steps. First we look at the case of a single transition. There is a single random variable, T, for the duration in the origin state until a transition to the destination state occurs. To derive a model, one assumes a known parametric distribution for this duration variable T, depending on some unknown parameters $a, b, c, \ldots$ The distribution may be given by a density function $f(t\,; a, b, c, \ldots)$ or, likewise, by a distribution function

$$F(t\,; a, b, c, \ldots) = \int_0^t f(\tau; a, b, c, \ldots)\, d\tau \tag{2}$$

or by a survivor function

$$G(t\,; a, b, c, \ldots) = 1 - F(t\,; a, b, c, \ldots) \tag{3}$$

or by a transition rate

$$r(t\,;a,b,c,\ldots) = \frac{f(t\,;a,b,c,\ldots)}{G(t\,;a,b,c,\ldots)} \tag{4}$$

Second, if episodes can end in one of several different destination states then, for each origin state $j \in \mathcal{O}$, there is a two-dimensional random variable (T_j, D_j), with T_j the duration in the origin state and D_j the destination state after leaving the origin state. Generalization of transition rate models to this case is based on the concept of transition-specific rates defined by

$$r_{jk}(t) = \lim_{\Delta t \to 0} \frac{\Pr\left(t \le T_j < t + \Delta t, D_j = k \mid T_j \ge t\right)}{\Delta t} \tag{5}$$

For the formulation of parametric transition rate models we assume that all transition-specific rate $r_{jk}(t)$ have the same mathematical form. In general, we begin by assuming a model for the single-transition case and simply extend this model to a situation with alternative destination states. The general approach is then

$$r_{jk}(t\,;a_{jk},b_{jk},c_{jk},\ldots) \equiv r(t\,;a_{jk},b_{jk},c_{jk},\ldots) \tag{6}$$

with $r(t\,;\ldots)$ taken from the single-transition case, but providing the possibility of transition-specific parameters.

To formulate the likelihood for the model one needs also survivor functions, $G_j(t)$, for the duration in the given origin state $j \in \mathcal{O}$. They can easily be derived from the transition-specific rates. One can first define transition-specific pseudo-survivor functions

$$\tilde{G}_{jk}(t) = \exp\left\{-\int_0^t r_{jk}(\tau)\,d\tau\right\} \tag{7}$$

As a consequence of assuming identical rates for the different transitions, all pseudo-survivor functions have the same mathematical form as the corresponding survivor function in the single-transition case, and $G_j(t)$ may be written as

$$G_j(t) = \prod_{k \in \mathcal{D}_j} \tilde{G}_{jk}(t) \tag{8}$$

Summing up: For the formulation of parametric transition rate models in TDA, we start with an expression for the rate in the single-transition case.

This is then generalized for the case of several transitions by assuming the same mathematical form for each transition-specific rate. Finally, the survivor function for the duration in a given origin states is derived from the transition-specific rates. As is seen in the next section, this is all what is needed for maximum likelihood estimation of the models.

There is, of course, a broad range of possibilities for the construction of transition rate models. One can start with some familiar parametric distribution and then derive the implied transition rate; or one can immediately start with some expression for the transition rates. Both approaches will be used in subsequent chapters. In any case, there are some unknown parameters, denoted by a_{jk}, b_{jk}, and c_{jk}, generally assumed to have different values for each transition. The models described in later chapters have, in fact, different numbers of unknown parameters (for each transition). There are simple models with only one set of parameters, and more flexible models with up to three different sets of parameters. Using the terminology of the RATE program (Tuma 1980), they are called, respectively, the A-terms, the B-terms, and the C-terms of a model.

The final step in model formulation is to specify how covariates shall be linked to the transition rate functions that primarily define the model. This, sometimes called the parameterization of a model, is done by linking covariates to the unknown parameters of the transition-specific rates. In principle, many different link functions are possible. Here we follow the most common approach: if parameters can only take positive values an exponential link function is used, otherwise a linear link function. The exponential link functions are

$$\begin{aligned} a_{jk} &= \exp\left\{A^{(j,k)}\,\alpha^{(j,k)}\right\} \\ b_{jk} &= \exp\left\{B^{(j,k)}\,\beta^{(j,k)}\right\} \\ c_{jk} &= \exp\left\{C^{(j,k)}\,\gamma^{(j,k)}\right\} \end{aligned} \tag{9}$$

$A^{(j,k)}$, $B^{(j,k)}$, and $C^{(j,k)}$, are (row) vectors of covariates, and in most cases it is assumed that the first component is equal to a constant one to provide an intercept term. $\alpha^{(j,k)}$, $\beta^{(j,k)}$, and $\gamma^{(j,k)}$, are the associated coefficients. Therefore, if no covariates are specified, TDA estimates a so-called *null model*, i.e. a model with only constant effects.

6.17.1.2 Maximum Likelihood Estimation

For model estimation TDA uses the method of maximum likelihood. To explain the setup of the likelihood, we proceed in three steps: first we consider the case of a single transition, then situations with one origin state but possibly two or more destination states, and finally situations with more than one origin state.

The notation to set up the likelihood is as follows. $\mathcal{O}$ is the set of origin states, and $\mathcal{D}_j$ is the set of possible destination states for episodes with origin $j \in \mathcal{O}$. $\mathcal{N}_j$ and $\mathcal{Z}_j$ are, respectively, the set of all episodes and the set of all right censored episodes, having an origin state j. $\mathcal{E}_{jk}$ is the set of all episodes with origin state j and destination state k which have an event ($j \neq k$). To simplify notation the dependence on parameters is not explicitly shown.

In the case of a single transition (j, k), the likelihood may be written as

$$\mathcal{L}_{jk} = \prod_{i \in \mathcal{E}_{jk}} f(t_i) \prod_{i \in \mathcal{Z}_j} G(t_i) = \prod_{i \in \mathcal{E}_{jk}} r(t_i) \prod_{i \in \mathcal{N}_j} G(t_i) \tag{1}$$

where $f(t)$ is the density function and $G(t)$ is the survivor function for the single transition (j, k). The contribution to the likelihood of an episode with an event at t_i is the density function, evaluated at the ending time t_i and with appropriate covariate values. The contribution of a censored episode is the survivor function evaluated at the censored ending time t_i, but possibly depending on covariates changing their values during the episode. Remember that, for the moment, we assume all episodes have starting time zero. The likelihood can be expressed, then, by using only the transition rate and the survivor function.

The next step is to look at the case with a single origin state j but possibly more than one destination state. The destination state space is $\mathcal{D}_j$. Using the notation introduced in **3.3.1**, the likelihood may be written as

$$\mathcal{L}_j = \prod_{k \in \mathcal{D}_j} \prod_{i \in \mathcal{E}_{jk}} \tilde{f}_{jk}(t_i) \prod_{i \in \mathcal{Z}_j} G_j(t_i) \tag{2}$$

The contribution of an episode with event at t_i is again the density function, but now transition-specific, according to the underlying model.

As shown in 3.3.1, this density is given by the subdensity function $\tilde{f}_{jk}(t)$, with the interpretation that $\tilde{f}_{jk}(t)\,dt$ is the probability of leaving the origin state j to the destination state k in a small time interval at t. The contribution to the likelihood of a right censored episode is the value of the survivor function at t_i, denoted by $G_j(t_i)$, i.e. the probability that the episode starting in state j has no event until t_i.

It is possible to factor the likelihood into a product of transition-specific terms.[1] First, the likelihood (2) may be rewritten as

$$\mathcal{L}_j = \prod_{k\in\mathcal{D}_j}\prod_{i\in\mathcal{E}_{jk}} r_{jk}(t_i) \prod_{i\in\mathcal{N}_j} G_j(t_i) \tag{3}$$

Secondly, using 6.17.1.1-(8), this may be rewritten as

$$\mathcal{L}_j = \prod_{k\in\mathcal{D}_j}\prod_{i\in\mathcal{E}_{jk}} r_{jk}(t_i) \prod_{k\in\mathcal{D}_j}\prod_{i\in\mathcal{N}_j} \tilde{G}_{jk}(t_i) \tag{4}$$

or, with terms rearranged:

$$\mathcal{L}_j = \prod_{k\in\mathcal{D}_j}\left\{\prod_{i\in\mathcal{E}_{jk}} r_{jk}(t_i) \prod_{i\in\mathcal{N}_j} \tilde{G}_{jk}(t_i)\right\} \tag{5}$$

The third step, to account for more than a single origin state, is again simple. One assumes that the model for each type of episode is conditional on a given origin state. Then also the total likelihood is the product of the likelihood for each of the origin states, $\mathcal{L}_j$, and can be written as

$$\mathcal{L} = \prod_{j\in\mathcal{O}}\prod_{k\in\mathcal{D}_j}\prod_{i\in\mathcal{E}_{jk}} r_{jk}(t_i) \prod_{i\in\mathcal{N}_j} \tilde{G}_{jk}(t_i) \tag{6}$$

In the case of a single origin state this likelihood reduces to (5), and in the case of a single transition it reduces to (1).

Estimation of parametric transition rate models in TDA is always based on the likelihood (6), by using the corresponding log-likelihood

$$\ell = \sum_{j\in\mathcal{O}}\sum_{k\in\mathcal{D}_j}\sum_{i\in\mathcal{E}_{jk}} \log\{r_{jk}(t_i)\} + \sum_{i\in\mathcal{N}_j} \log\left\{\tilde{G}_{jk}(t_i)\right\} \tag{7}$$

[1] This assumes transition rates defined in continuous time. For discrete-time models this factoring is not possible as will be discussed in 6.17.6.

The fact that the total likelihood can be expressed as a product of transition-specific factors implies that, in the case of several transitions, a model for each transition can be estimated separately.[2] Estimation for a transition (j, k) is then done using all episodes starting in origin state j; the episodes ending in destination state k are regarded as having an event, and *all other* episodes are included as censored.

The possibility of factoring the total likelihood opens an easy way to estimate models with a different specification of transition rates for different transitions. A different model can be assumed for each transition, and then each of these models can be estimated separately.

[2]Of course, this is only possible if there are no constraints on parameters across different transitions. To provide for this possibility is the main reason for using the likelihood (6).

6.17.1.3 Time-varying Covariates

For practical applications one should be able to include time-varying covariates into the specification of transition rate models. There are basically three different approaches.

1. The first approach develops a special model formulation that accounts for time-dependent covariates. For each application, this method requires a new formulation of the model and, accordingly, new equations for maximum likelihood estimation must be derived. This is impossible with a general purpose program and not supported by TDA.

2. A second method was proposed by Tuma (1980). The idea is to split the time axis into predefined time intervals and then to specify possibly different covariates with associated coefficients for each of these intervals. This method is supported by one version of the piecewise-constant exponential model discussed in **6.17.2.3**.

3. In practice, the most useful method is known as *episode splitting*, and this is the preferred method for handling time-varying covariates in TDA. The basic idea is very simple. At all time points when at least one of the covariates changes its value, the original episode is split into subepisodes, called splits (of an episode), and each split is given appropriate values of the covariates. All splits have the same origin state as the original episode and, except the last split, are regarded as right censored. Only the last split is given the same destination state as the original episode. To be used with split episodes, the likelihood expression for transition rate models needs some modification. We need an expression based on split episodes that might have starting times greater than zero. Fortunately, the necessary modifications in the likelihood construction are simple. Consider one of the original episodes (s_i, t_i), with $s_i = 0$, and assume it is split into l_i sub-episodes

$$(s_i, t_i) \equiv \{(s_i^{(l)}, t_i^{(l)}) \mid l = 1, \ldots, l_i\} \tag{1}$$

The pseudo-survivor functions defined in (7) can then be written as

$$\tilde{G}_{jk}(t_i) = \prod_{l=1}^{l_i} \tilde{G}_{jk}(t_i^{(l)} \mid s_i^{(l)}) \tag{2}$$

where, on the right-hand side, one has conditional pseudo-survivor functions[1] defined by

$$\tilde{G}_{jk}(t_i^{(l)} \mid s_i^{(l)}) = \frac{\tilde{G}_{jk}(t_i^{(l)})}{\tilde{G}_{jk}(s_i^{(l)})} = \exp\left\{ -\int_{s_i^{(l)}}^{t_i^{(l)}} r_{jk}(\tau)\, d\tau \right\} \tag{3}$$

In the integral on the right-hand side, $r_{jk}(\tau)$ is the rate appropriate for the split $(s_i^{(l)}, t_i^{(l)})$ and may depend on the covariate values for this split.

Now it would be possible to rewrite again the likelihood function (6) by inserting (3). But one does not really need the subscripts, l. One can simply write the likelihood function as

$$\mathcal{L} = \prod_{j \in \mathcal{O}} \prod_{k \in \mathcal{D}_j} \prod_{i \in \mathcal{E}_{jk}} r_{jk}(t_i) \prod_{i \in \mathcal{N}_j} \tilde{G}_{jk}(t_i \mid s_i) \tag{4}$$

Written this way, the likelihood can be used as well with a sample of original (not split) episodes where all starting times are zero, and with a sample of splits. Of course, it is the responsibility of the user to do any episode splitting in such a way that the splits add up to a sample of meaningful episodes.

[1] In the case of a single transition it is the probability of staying in the origin state at least until $t_i^{(l)}$ given that this state was not exited until $s_i^{(l)}$.

6.17.1.4 The rate Command

TDA's command to request estimation of a transition rate model is **rate**. The syntax is

```
rate (parameter) = model_number;
```

Currently available model numbers are shown in Box 1; parameter options are summarized in Box 2. Also, most options available in the **fmin** command for function minimization can be use with the **rate** command, see **5.6.1**. These additional options are not shown in Box 2.

1. For most models all parameters are optional and the **rate** command can be used as

```
rate = model_number;
```

resulting in the estimation of a null model withou covariates. An exception are the piecewise constant exponential models where it is required to use the **tp** parameter to define time periods.

2. Possibilities to include covariates depend on the model terms provided by the selected model (see **6.17.1.1**). In general, covariates are linked to model terms by using expressions like

```
xa(j,k) = list_of_variables,
```

meaning that the list of variables to be given on the right-hand side is linked to transition from origin state j to destination state k in the model term A. If a model also provides a model term B, one can use analoguously the expression

```
xb(j,k) = list_of_variables,
```

Some models also provide model terms C and D. The available possibilities depend on the type of model and will be explained separtely for each model.

3. For some models, additional parameters can be used for model specification. If a model contains polynomial terms, one can use the **deg**

Box 1 Models available with the `rate` command

		Model
rate =	1	Cox model
	2	Exponential model
	3	Piecewise constant exponential model
	4	Polynomial rates, I
	5	Polynomial rates, II
	6	Gompertz-Makeham models
	7	Weibull model
	8	Sickle model
	9	Log-logistic model, type I
	10	Log-logistic model, type II
	12	Log-normal model
	13	Generalized gamma model
	14	Inverse Gaussian model
	16	Piecewise constant exponential model with period-specific effects
	20	Logistic regression model
	21	Complementary log-log model

parameter to specify the degree. The `kgam` parameter is used for the generalized gamma model. The `grp` parameter is used to define groups for stratified Cox model estimation. Additional explanations for these parameters will be given in separate sections.

4. Basic results of model estimation are always written into the standard output. This includes information about the results of maximum likelihood estimtion, in particular whether the selected minimization algorithm has reached convergence. In any case, one gets a table with estimated parameters and, if the algorithm terminated successfully, also estimated standard errors. The print format can be controlled with the `tfmt` option, default is `tfmt=10.4`. In addition, for some models, one can use the `rrisk` option to request a table containing relative risks; see 6.17.1.5.

5. The `ppar` option can be used to request an output file containing the estimated parameters and standard errors. The syntax is

```
ppar = name_of_an_output_file,
```

Box 2 Syntax for `rate` command

```
rate (
     tp=...,          time periods
     xa(j,k)=...,     variables for A-term of model
     xb(j,k)=...,     variables for B-term of model
     xc(j,k)=...,     variables for C-term of model
     xd(j,k)=...,     variables for D-term of model
     deg=...,         degree of polynomial rate, def. 0
     kgam=...,        coefficient for gamma model, def. 1
     mix=...,         selection of mixture model, see 6.17.4.1
     grp=...,         groups for stratified Cox model
     rrisk,           print table with relative risks
     ppar=...,        output file with estimated parameters
     pcov=...,        output file with covariance matrix
     pres=...,        generalized residuals, see 6.17.1.6
     prate=...,       calculation of estimated rates
     tfmt=...,        print format for results, def. 10.4
     mfmt=...,        print format for pcov, pres and prate, def. 12.4
     mplog=...,       write loglikelihood value into matrix
     mppar=...,       write estimated parameters into matrix
     mpcov=...,       write covariance matrix into matrix
     mpgrad=...,      write gradients into matrix
     dsv=...,         input file with starting values
     con=...,         linear parameter constraints
                      in addition, one can use all parameters to
                      control maximum likelihood estimation.
) = model_number;
```

The print format is the same as used for the table in the standard output and can be changed with the `tfmt` option.

6. As mentioned, standard errors are only calculated if the minimization algorithm terminates successfully. In this case it is also possible to request an output file containing the full covariance matrix with the `pcov` option. The print format can be controlled with the `mmfmt` option, default is `mfmt=12.4`.

7. For many models TDA can calculate so-called generalized residuals, see 6.17.1.6. If supported, one can use the option

```
pres = name_of_an_output_file,
```

to request an output file containing the estimated generalized residuals. The print format can be controlled with the `mfmt` option, default is `mfmt=12.4`.

8. One often wishes to tabulate, or plot, the transition rates, survivor and density functions resulting from estimation of a model, depending on certain constellations of covariates. This can be done with the `prate` option, the syntax is

```
prate (tab=a(d)b, V1=..., V2=...,...) = fname,
```

The right-hand side must provide the name of an output file. Except for the Cox model (see below), the `tab` parameter must be used to specify a time axis, running from `a` to `b` with increments `d`; `a` must be nonnegative, `b` must be greater than `a`, and `d` must be positive. Then follows, optionally, a list of variable names each with syntax

```
VName = v,
```

where `VName` is the name of a variable used in the model specification and v is the value to be used for this variable when estimating rates. All variable not explicitly mentioned will get the value zero. Consider, for example, the expression

```
prate (tab=0(1)100, X1=1, X2=2) = r.dat,
```

The output file, `r.dat`, will tabulate the estimated rates for the time axis $0, 1, 2, \ldots, 100$ and uses covariate values `X1=1` and `X2=2`; all other covariates (if any) will be zero. We mention that the `prate` option can be used more than once in the same `rate` command. Each `prate` expression will create a new table. If the output file name is the same as previously used, the table will be appended to the end of the file.

For the Cox model, the `prate` option can be used to request tables with estimated baseline rates; this will be explained in **6.17.7.6**. For other parametric models, the output file will contain the estimated rates, survivor and density functions. The calculation depends on the type of model and will be described separately for each type of model. The print format can be controlled with the `mfmt` option, default is `mfmt=12.4`. For an example, see **6.17.2.1**.

9. The `dsv` parameter can be used to provide starting values for model estimation. The syntax is

```
dsv = name_of_an_input_file,
```

The input file is assumed to be a free-format file. TDA uses the first numerical entry in each record assuming an ordering of parameters as used in model estimation and shown in the table of estimated parameters. In particular, it is possible to use an output file created with the `ppar` option as an input file for the `dsv` option. Alternatively, one can use the `xp` parameter to provide starting values, see **5.6.4**.

10. The `con` option can be used to specify linear constraints for the model parameters, see the discussion in **5.6**. Also most other options, discussed there, to control maximum likelihood estimation can be used as additional parameters in the `rate` command.

11. For a few parametric rate models one can use the `mix` parameter to request model estimation based on a gamma mixing distribution, see the discussion in **6.17.4.1**.

6.17.1.5 Relative Risks

Many of TDA's transition rate models can be written as proportional rate models:

$$r(t) = r_b(t;\theta)\ \exp\left(\sum_{i=1}^{m} A_i\alpha_i\right) \tag{1}$$

Here the transition rate, $r(t)$, is a product of two parts. The first part, $r_b(t;\theta)$, is called the *baseline* transition rate. In general, with the exception of the simple exponential model, the baseline rate depends on time and on a parameter vector θ. The second part links the transition rate to individual-specific covariates; the formulation in (1) assumes m covariates, $A_1, \ldots, A_m$, with associated parameters $\alpha_1, \ldots, \alpha_m$.

Models which can be given the representation (1) are called *proportional* rate models because the effects of individual-specific covariates are independent of time. Only the baseline transition rate is dependent on time; different individual-specific covariates only lead to proportional shifts of the baseline rate.

Of course, this interpretations assumes that the baseline rate's parameter vector, θ, is not also linked to individual-specific covariates. In fact, most transition rate models which can be estimated with TDA also provide the opportunity to link some or all components of θ to individual-specific covariates; and if this is done the resulting model is no longer a proportional rate model.

The main advantage of proportional rate models is their simple interpretation. There is a single baseline rate, independent of individual-specific covariates and therefore easily assessed, for instance by plotting this rate. Then, having a picture of the baseline rate, the influence of individual-specific covariates can be represented by their contribution to shifting the baseline rate. This is easily seen by rewriting (1) as

$$r(t) = r_b(t;\theta)\ \prod_{i=1}^{m} r_i(A_i) \quad \text{with} \quad r_i(A_i) = \exp(A_i\alpha_i) \tag{2}$$

For instance, if a_1 and a_2 are two values of the covariate A_i, the proportional shift of the baseline rate due to a change from a_1 to a_2 is

$r_i(a_2)/r_i(a1)$. In particular, if A_i is a dummy variable with possible values 0 and 1, the whole information about this variable's influence on the transition rate can be given by $r_i(1)$, i.e. by $\exp(\alpha_i)$.

To aid in this interpretation of proportional transition rate models, TDA provides the option to request a table containing the estimated relative risks, $r_i(1)$. To request this table, simply use `rrisk` as a parameter in the `rate` command. Note, however, that for some transition rate models, this option is not available. Whether it is, and how the calculations are performed, is described separately for each type of model.

6.17.1.6 Generalized Residuals

Since estimation of transition rate models is based on maximum likelihood, it is difficult to assess goodness of fit. As proposed by Cox and Snell [1968], it might be helpful to calculate so-called generalized residuals. For applying this idea to transition rate models see, for instance, Blossfeld et al. [1989, p. 82] and Lancaster [1985].

The definition is as follows. Let $\hat{r}(t;x)$ denote the estimated rate, depending on time t and on a vector of covariates, x. The estimation is based on a random sample of individuals $i = 1,\ldots,N$, with duration t_i and covariate vectors x_i. Generalized residuals are then defined as cumulative transition rates, evaluated for the given sample observations, i.e.

$$\hat{e}_i = \int_0^{t_i} \hat{r}(\tau; x_i)\, d\tau \qquad i = 1,\ldots,N \tag{1}$$

The reasoning behind this definition is that, if the model is appropriate, and if there were no censored observations, the set of residuals should approximately behave like a sample from a standard exponential distribution. If some of the observations are censored, the residuals may be regarded as a censored sample from a standard exponential distribution. In any case, one can calculate a product-limit estimate of the survivor function of the residuals, say $G_{\hat{e}}(e)$, and a plot of $-\log(G_{\hat{e}}(\hat{e}_i))$ against $\hat{e}_i$ can be used to check whether the residuals actually follow a standard exponential distribution.

For most of TDA's continuous time transition rate models one can request a table with generalized residuals by using the option

```
pres = name_of_an_output_file,
```

as part of the `rate` command for model estimation. Depending on the estimated model, residuals are calculated and then written into the specified output file. The print format can be controlled with the `mfmt` option, default is `mfmt=12.4`. The variables (columns) in the output file are as follows:

1. Case number.
2. Origin state of the episode.

3. Destination state of the episode.
4. Starting time of the episode.
5. Ending time of the episode.
6. Estimated transition rate, evaluated at the ending time of the episode. Note that the calculation is independent of whether the case is right censored or not. The information about origin and destination states can be used to select censored or uncensored cases.
7. Estimated survivor function, evaluated at the ending time of the episode.
8. Residual.
9. Case weight.

Calculation of residuals is conditional on given starting times which may be greater than zero. There will then be no straightforward interpretation. In any case, residuals are calculated according to

$$\hat{e}_i = \int_{s_i}^{t_i} \hat{r}(\tau; x_i)\, d\tau \qquad i = 1, \ldots, N \tag{2}$$

with s_i and t_i denoting the starting and ending time of the episode, respectively. Also the survivor functions printed into the output file are conditional on starting times, meaning that the program calculates and prints

$$\hat{G}(t_i \mid s_i; x_i) = \exp\left\{ - \int_{s_i}^{t_i} \hat{r}(\tau; x_i)\, d\tau \right\} \qquad i = 1, \ldots, N \tag{3}$$

Therefore, if the starting times are not zero, in particular if the method of episode splitting is applied, the output file will not contain proper information about residuals. We also mention that calculation of generalized residuals is only supported for single episode data without alternative destination states.[1] It is also not supported for Cox models, mixture models, and discrete time transition rate models.

[1] If one really needs these residuals for models with alternative destination states one should estimate a separate model for each transition.

6.17.2 Exponential Models

This section describes transition rate models with a constant, or piecewise constant, rate. The sections are:

6.17.2.1 The Basic Exponential Model

The most simple transition rate model is the exponential where it is assumed that the transition rates can vary with different constellations of covariates but do not depend on time. In the case of a single transition the model is derived by assuming that the duration variable T can be described by an exponential distribution with density, survivor function, and transition rates, respectively, given by

$$f(t) = a\exp(-a\,t) \qquad a > 0 \tag{1}$$

$$G(t) = \exp(-a\,t) \tag{2}$$

$$r(t) = a \tag{3}$$

The transition rates are constants, not varying with the duration of the episode. In most applications this might be not realistic. But the model is not only simple, it can serve as a reference for more complicated models, and some more flexible extensions of this model are described in later sections.

Implementation. The simple exponential model has model number 2; the command to request model estimation is

```
rate (parameter) = 2;
```

For a description of the syntax and options see **6.17.1.4**.

The exponential distribution has only one parameter, so there is only one possibility to include covariates (there is only an A-term in this model). TDA uses an exponential link function. Taking into account the possibility of different transitions one gets the model formulation

$$r_{jk} = \exp\left\{A^{(jk)}\alpha^{(jk)}\right\} \tag{4}$$

r_{jk} is the constant transition rate from origin state j to destination state k. $A^{(jk)}$ is a (row) vector of covariates, having a constant one in its first component to provide an intercept term; and $\alpha^{(jk)}$ is a vector of associated coefficients to be estimated.

The command `rate=2` estimates a null model without covariates. For linking covariates to the model for transition from j to k one should use

the parameter

```
xa(j,k) = list_of_variables,
```

Maximum Likelihood Estimation. The model is estimated following the outline given in **6.17.1.2** and **6.17.1.3**. To simplify notation we omit indices for transitions. Using the conditional survivor function

$$G(t_i \mid s_i) = \exp\left\{-(t_i - s_i)\exp(A_i\,\alpha)\right\} \tag{5}$$

the log-likelihood is

$$\ell = \sum_{i\in\mathcal{E}} A_i\alpha + \sum_{i\in\mathcal{N}} (s_i - t_i)\ \exp(A_i\,\alpha)$$

Using the same notation, the first and second derivatives of the log-likelihood with respect to the α coefficients are derived to be

$$\begin{aligned} \frac{\partial \ell}{\partial \alpha_{j_a}} &= \sum_{i\in\mathcal{E}} A_{i,j_a} + \sum_{i\in\mathcal{N}} (s_i - t_i)\exp(A_i\alpha)A_{i,j_a} \qquad (6)\\ \frac{\partial^2 \ell}{\partial \alpha_{j_a}\partial \alpha_{k_a}} &= \sum_{i\in\mathcal{N}} (s_i - t_i)\exp(A_i\alpha)A_{i,j_a}A_{i,k_a} \end{aligned}$$

The notation $A_{i,j}$ is used to denote the jth component of the covariate vector A_i associated with the ith episode.

Starting Values. In most cases there are no difficulties in reaching a ML solution for the exponential model. The calculation of default starting values by TDA is build on the fact that the ML solution of an exponential null model can be found directly and does not need an iterative procedure.[1] In the single transition case the rate of an exponential null model is given by

$$r_n = \frac{W_u}{T_w} \tag{7}$$

where W_u is the number of uncensored episodes, and T_w is the summed duration of *all* episodes. Default starting values for exponential models are calculated as the parameters of a corresponding null model.

[1] Cf. Lawless [1982, p. 101].

Box 1 Command file `rt1.cf` for simple exponential model

```
nvar(
    dfile = rrdat.1,   # data file

    ID    [3.0] = c1,  # identification number
    SN    [2.0] = c2,  # spell number
    TS    [3.0] = c3,  # starting time
    TF    [3.0] = c4,  # ending time
    SEX   [2.0] = c5,  # sex (1 men, 2 women)
    TI    [3.0] = c6,  # interview date
    TB    [3.0] = c7,  # birth date
    TE    [3.0] = c8,  # entry into labor market
    TMAR  [3.0] = c9,  # marriage date (0 if no marriage)
    PRES  [3.0] = c10, # prestige of current job
    PRESN [3.0] = c11, # prestige of next job
    EDU   [2.0] = c12, # highest educational attainment

    COHO1 = ge(TB,348) & le(TB,384), # birth cohort 1
    COHO2 = ge(TB,468) & le(TB,504), # birth cohort 2
    COHO3 = ge(TB,588) & le(TB,624), # birth cohort 3
    W     = SEX[2],                  # women = 1

    DES [1.0] = if eq(TF,TI) then 0 else 1,  # destination state
    DUR [3.0] = TF - TS + 1,
);

edef(
    ts = 0,     # starting time
    tf = DUR,   # ending time
    org = 0,    # origin state
    des = DES,  # destination state
);

rate (
    xa(0,1) = COHO2,COHO3,W,

) = 2;      exponential model
```

Example 1 To illustrate estimation of an exponential model we use our main example data, `rrdat.1` (see **3.3.3**). Box 1 shows the command file, `rt1.cf`. Most commands are used to specify variables and episode data. The `rate` command at the end of the file requests model estimation. Three covariates are linked to the model for transition from 0 to 1, the only transition in the input data. For all other options of the `rate` command, default values are used. The output from the `rate` command is shown in Box 2. Convergence is reached with 5 iterations using TDA's

Box 2 Output from command file `rt1.cf`

```
Transition rate models.
Model: Exponential

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Number of model parameters: 4
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Log-likelihood of exponential null model: -2514.02
Scaling factor for log-likelihood: -0.001
Using default starting values.

Convergence reached in 5 iterations.
Number of function evaluations: 6 (6,6)

Maximum of log likelihood: -2475.44
Norm of final gradient vector: 9.66315e-08
Last absolute change of function value: 2.62535e-11
Last relative change in parameters: 3.26799e-05

Idx SN Org Des MT Variable      Coeff      Error    C/Error  Signif
-------------------------------------------------------------------
  1  1   0   1  A Constant    -5.0114     0.0843   -59.4446  1.0000
  2  1   0   1  A COHO2        0.5341     0.1120     4.7686  1.0000
  3  1   0   1  A COHO3        0.6738     0.1152     5.8472  1.0000
  4  1   0   1  A W            0.5065     0.0942     5.3746  1.0000

Log likelihood (starting values): -2514.0201
Log likelihood (final estimates): -2475.4383
```

default algorithm for ML estimation. The table shows the estimated parameter values and standard errors.

Example 2 If the input data contain several transitions covariates can be linked separately to each transition. To illustrate this option we use command file `rt1m.cf`, shown in Box 4. There are now three transitions, and we link some covariates to each of them. While it would be possible to use different sets of covariates for each transition, we have not done

Box 3 Command file `rt1m.cf`

```
nvar(
    dfile = rrdat.1,   # data file

    ID    [3.0] = c1,  # identification number
    SN    [2.0] = c2,  # spell number
    TS    [3.0] = c3,  # starting time
    TF    [3.0] = c4,  # ending time
    SEX   [2.0] = c5,  # sex (1 men, 2 women)
    TI    [3.0] = c6,  # interview date
    TB    [3.0] = c7,  # birth date
    TE    [3.0] = c8,  # entry into labor market
    TMAR  [3.0] = c9,  # marriage date (0 if no marriage)
    PRES  [3.0] = c10, # prestige of current job
    PRESN [3.0] = c11, # prestige of next job
    EDU   [2.0] = c12, # highest educational attainment

    COHO1 = ge(TB,348) & le(TB,384), # birth cohort 1
    COHO2 = ge(TB,468) & le(TB,504), # birth cohort 2
    COHO3 = ge(TB,588) & le(TB,624), # birth cohort 3
    W     = SEX[2],                  # women = 1

    DES [1.0] = if eq(TF,TI) then 0 else
                if ge(PRESN/PRES - 1,0.2) then 1 else
                if lt(PRESN/PRES - 1,0.0) then 3 else 2,

    DUR [3.0] = TF - TS + 1,
);

edef(
    ts = 0,     # starting time
    tf = DUR,   # ending time
    org = 0,    # origin state
    des = DES,  # destination state
);

rate (
    xa(0,1) = COHO2,COHO3,W,
    xa(0,2) = COHO2,COHO3,W,
    xa(0,3) = COHO2,COHO3,W,

) = 2;      exponential model
```

so in this example. Part of TDA's standard output from this command file is shown in Box 4.

Example 3 To control model estimation, one can use all the options

Box 4 Part of standard output from file `rt1m.cf`

```
Convergence reached in 5 iterations.
Number of function evaluations: 6 (6,6)

Maximum of log likelihood: -2935.58
Norm of final gradient vector: 2.6882e-05
Last absolute change of function value: 6.16377e-09
Last relative change in parameters: 0.000460972

Idx SN Org Des MT Variable      Coeff      Error    C/Error  Signif
-----------------------------------------------------------------
  1  1   0   1  A Constant    -6.3412     0.1736   -36.5229  1.0000
  2  1   0   1  A COHO2        0.3012     0.2618     1.1504  0.7500
  3  1   0   1  A COHO3        0.4999     0.2651     1.8857  0.9407
  4  1   0   1  A W           -0.0957     0.2342    -0.4084  0.3170
-----------------------------------------------------------------
  5  1   0   2  A Constant    -5.7548     0.1236   -46.5439  1.0000
  6  1   0   2  A COHO2        0.6924     0.1630     4.2486  1.0000
  7  1   0   2  A COHO3        0.8088     0.1684     4.8039  1.0000
  8  1   0   2  A W            0.2977     0.1381     2.1556  0.9689
-----------------------------------------------------------------
  9  1   0   3  A Constant    -6.3686     0.1578   -40.3698  1.0000
 10  1   0   3  A COHO2        0.4424     0.1924     2.2996  0.9785
 11  1   0   3  A COHO3        0.5875     0.1980     2.9676  0.9970
 12  1   0   3  A W            1.1156     0.1659     6.7254  1.0000

Log likelihood (starting values): -2986.9500
Log likelihood (final estimates): -2935.5833
```

described in 5.6.1. For example, one can use the `ppar` option to write estimated model parameters into an output file, the `pcov` option for the covariance matrix, and the `dsv` option for reading starting values.

It is also possible to estimate models with linear constraints on a subset of parameters. Assume, for example, we want to estimate the model defined in command file `rt1m.cf` with the additional constraint that the effects of `COHO3` are equal for the transitions $(0, 1)$ and $(0, 3)$. The corresponding model parameters have indices 3 and 11, respectively. So the additional parameter for the `rate` command should be

```
con = b3 - b11 = 0,
```

Relative Risks. The `rrisk` option can be used as part of the `rate` command to request a table with relative risks. Since the exponential model is a proportional transition rate model, the estimated model parameters

Box 5 Relative risks for the model estimated by command file `rt1m.cf`

Idx	SN	Org	Des	MT	Variable	R.Risk
1	1	0	1	A	Constant	0.0018
2	1	0	1	A	COHO2	1.3514
3	1	0	1	A	COHO3	1.6485
4	1	0	1	A	W	0.9088
5	1	0	2	A	Constant	0.0032
6	1	0	2	A	COHO2	1.9985
7	1	0	2	A	COHO3	2.2452
8	1	0	2	A	W	1.3468
9	1	0	3	A	Constant	0.0017
10	1	0	3	A	COHO2	1.5564
11	1	0	3	A	COHO3	1.7994
12	1	0	3	A	W	3.0514

Box 6 Generalized residuals estimated with `rt1.cf`

# Case	Org	Des	TS	TF	Rate	Survivor Function	Residual	Weight
1	0	0	0.0	428.0	0.0067	0.0578	2.8513	1.0000
2	0	1	0.0	46.0	0.0111	0.6014	0.5086	1.0000
3	0	1	0.0	34.0	0.0111	0.6867	0.3759	1.0000
4	0	1	0.0	220.0	0.0111	0.0878	2.4322	1.0000
5	0	1	0.0	12.0	0.0189	0.7975	0.2263	1.0000
...								
595	0	1	0.0	119.0	0.0067	0.4526	0.7928	1.0000
596	0	1	0.0	66.0	0.0067	0.6442	0.4397	1.0000
597	0	0	0.0	112.0	0.0067	0.4742	0.7461	1.0000
598	0	0	0.0	103.0	0.0131	0.2603	1.3461	1.0000
599	0	1	0.0	22.0	0.0189	0.6604	0.4149	1.0000
600	0	1	0.0	13.0	0.0189	0.7826	0.2452	1.0000

can be changed into relative risks by taking their anti-logarithm. Box 5 shows the effect of the `rrisk` option when added to the `rate` command in command file `rt1m.cf` (Box 3).

Generalized Residuals. If the exponential model is estimated for only a single transition, one can use the `pres` option to request a table with generalized residuals. For each episode $(o_i, d_i, s_i, t_i, A_i)$ in the input data, the output file specified with the `pres` option will contain the estimated

rate

$$\hat{r}_i = \exp(A_i\,\hat{\alpha})$$

the estimated (conditional) survivor function

$$\hat{G}(t_i \mid s_i) = \exp(-(t_i - s_i)\,\hat{r}_i)$$

and the generalized residual

$$\hat{e}_i = -\log(\hat{G}(t_i \mid s_i))$$

To illustrate the structure of the output file, the parameter `pres=res.d` is added to the `rate` command in command file `rt1.cf`. Part of the resulting output file, `res.d`, is shown in Box 6.

Box 7 Table with estimated rates based on `rt1.cf`

```
# Time axis: 0 (5) 100

# Idx SN Org Des MT Variable      Coeff  Covariate
# ------------------------------------------------
#   1  1   0   1  A Constant    -5.0114     1.0000
#   2  1   0   1  A COHO2        0.5341     0.0000
#   3  1   0   1  A COHO3        0.6738     1.0000
#   4  1   0   1  A W            0.5065     1.0000

# ID          Time       Surv.F      Density         Rate
# ---------------------------------------------------------
   0        0.0000       1.0000       0.0217       0.0217
   0        5.0000       0.8972       0.0195       0.0217
   0       10.0000       0.8050       0.0175       0.0217
   0       15.0000       0.7223       0.0157       0.0217
   0       20.0000       0.6481       0.0141       0.0217
   0       25.0000       0.5815       0.0126       0.0217
   0       30.0000       0.5217       0.0113       0.0217
   0       35.0000       0.4681       0.0102       0.0217
   0       40.0000       0.4200       0.0091       0.0217
   0       45.0000       0.3768       0.0082       0.0217
   0       50.0000       0.3381       0.0073       0.0217
   0       55.0000       0.3034       0.0066       0.0217
   0       60.0000       0.2722       0.0059       0.0217
   0       65.0000       0.2442       0.0053       0.0217
   0       70.0000       0.2191       0.0048       0.0217
   0       75.0000       0.1966       0.0043       0.0217
   0       80.0000       0.1764       0.0038       0.0217
   0       85.0000       0.1583       0.0034       0.0217
   0       90.0000       0.1420       0.0031       0.0217
   0       95.0000       0.1274       0.0028       0.0217
   0      100.0000       0.1143       0.0025       0.0217
```

Printing Estimated Rates. The `prate` option can be used to request tables with estimated rates, survivor and density functions, depending on selected covariate constellations. To illustrate this option, we use command file `rt1.cf` and add the option

```
prate (tab=0(5)100, COHO3=1,W=1) = r.dat,
```

to the `rate` command. The resulting output file, `r.dat`, is shown in Box 7. The header shows the parameter and covariate values used to create the table. The calculation of the rate, survivor and density functions is based on the estimated model parameters and the specified covariate values using the formulas for the exponential model. The first column of

each table, labelled by `ID`, can be used to identify the table. If the `prate` option is used more than once in the same `rate` command then, using the same output file name, it will contain several tables which can be identified by their `ID` column.

6.17.2.2 Models with Time Periods

For most applications, the basic exponential model is not appropriate. However, a small generalization leads to a very useful and flexible model, called the piecewise constant exponential model. The idea is to split the time axis into periods and assume that transition rates are constant in each period but may vary across periods.

If there are m time periods, the distribution of durations has m parameters. To include covariates one has two different options. The first is to assume that only a baseline rate, given by period-specific constants, can vary across time periods but the covariates have the same (proportional) effects in each period. The second option would be to allow for period-specific effects of covariates. TDA's standard exponential model with time periods follows the first option; a model with period specific effects is described in **6.17.2.3**.

Implementation. The exponential model with time periods has model number 3; the command to request model estimation is

```
rate (tp=..., additional parameters) = 3;
```

The `tp` parameter must be used to provide time periods defined by split points on the time axis:

$$0 = \tau_1 < \tau_2 < \tau_3 < \ldots < \tau_m \tag{1}$$

With $\tau_{m+1} = \infty$, one gets m time periods

$$I_l = \{t \mid \tau_l \leq t < \tau_{l+1}\} \qquad l = 1, \ldots, m \tag{2}$$

The syntax of the `tp` parameter is

`tp =` $\tau_1, \tau_2, \ldots, \tau_m$`,`

This parameter must be used with $\tau_1 = 0$ when requesting an exponential model with time periods.

Given these time periods, the transition rate from origin state j to destination state k is

$$r_{jk}(t) = \exp\left\{\bar{\alpha}_l^{(jk)} + A^{(jk)}\alpha^{(jk)}\right\} \quad \text{if} \quad t \in I_l \tag{3}$$

For each transition (j,k), $\bar{\alpha}_l^{(jk)}$ is a constant coefficient associated with the lth time period. $A^{(jk)}$ is a (row) vector of covariates, and $\alpha^{(jk)}$ is an associated vector of coefficients assumed not to vary across time periods. The model must not contain a separate constant. If specified without covariates, TDA estimates a null model, containing only parameters for the baseline rate. Covariates can be added to the A-term of the model with the option

```
xa(j,k) = list_of_variables,
```

Maximum Likelihood Estimation. The model is estimated following the outline given in **6.17.1.2** and **6.17.1.3**. To simplify notation we omit indices for transitions and define $l[t]$ to be the index of the time period containing t (so always $t \in I_{l[t]}$). Also the following notation is helpful:

$$\delta[t,l] = \begin{cases} 1 & \text{if} \quad t \in I_l \\ 0 & \text{else} \end{cases} \tag{4}$$

$$\Delta[s,t,l] = \begin{cases} t-\tau_l & \text{if} \quad s \le \tau_l,\ \tau_l < t < \tau_{l+1} \\ \tau_{l+1}-\tau_l & \text{if} \quad s \le \tau_l,\ t \ge \tau_{l+1} \\ \tau_{l+1}-s & \text{if} \quad t \ge \tau_{l+1},\ \tau_l < s < \tau_{l+1} \\ 0 & \text{else} \end{cases} \tag{5}$$

The conditional survivor function may then be written as

$$G(t \mid s) = \exp\left\{ -\sum_{l=1}^{m} \Delta[s,t,l] \exp(\bar{\alpha}_l + A\alpha) \right\} \tag{6}$$

Using this expression, the log-likelihood can be written as

$$\ell = \sum_{i \in \mathcal{E}} (\bar{\alpha}_{l[t_i]} + A_i\alpha) - \sum_{i \in \mathcal{N}} \sum_{l=1}^{m} \Delta[s_i,t_i,l] \exp(\bar{\alpha}_l + A_i\alpha) \tag{7}$$

and the first and second derivatives are[1]

$$\frac{\partial \ell}{\partial \bar{\alpha}_j} = \sum_{i \in \mathcal{E}} \delta[t_i,j] + \sum_{i \in \mathcal{N}} -\Delta[s_i,t_i,j] \exp(\bar{\alpha}_j + A_i\alpha)$$

$$\frac{\partial \ell}{\partial \alpha_j} = \sum_{i \in \mathcal{E}} A_{i,j} + \sum_{i \in \mathcal{N}} \sum_{l=1}^{m} -\Delta[s_i,t_i,l] \exp(\bar{\alpha}_l + A_i\alpha)\, A_{i,j}$$

[1] δ_{jk}, here and in later sections, denotes the Kronecker symbol which is equal to one, if $j = k$, and is otherwise zero.

$$\frac{\partial^2 \ell}{\partial \bar{\alpha}_j \partial \bar{\alpha}_k} = \sum_{i \in \mathcal{N}} -\Delta[s_i, t_i, j] \exp(\bar{\alpha}_j + A_i \alpha)\, \delta_{jk}$$

$$\frac{\partial^2 \ell}{\partial \alpha_j \partial \alpha_k} = \sum_{i \in \mathcal{N}} \sum_{l=1}^{m} -\Delta[s_i, t_i, l] \exp(\bar{\alpha}_l + A_i \alpha) A_{i,j} A_{i,k}$$

$$\frac{\partial^2 \ell}{\partial \bar{\alpha}_j \partial \alpha_k} = \sum_{i \in \mathcal{N}} -\Delta[s_i, t_i, j] \exp(\bar{\alpha}_j + A_i \alpha) A_{i,k}$$

Starting Values. TDA calculates default starting values for the piecewise constant exponential model by assuming the same rate in each time period. As an initial estimate the constant rate of a correspondingly defined exponential null model is used. In most practical situations this is sufficient to reach convergence of the maximum likelihood iterations. Alternatively, one can use the `dsv` option to provide starting values.

Example 1 To illustrate estimation of an exponential model with time periods we continue with command file `rt1.cf` (see 6.17.2.1), and modify the `rate` command. The model number is now 3, instead of 2; and we add the parameter

```
tp = 0 (12) 96,
```

to specify time periods. The new command file is `rt2.cf` (not shown). The output from this command is shown in Box 1. It begins with a table showing the time periods and the distribution of starting and ending times of the episodes used for model estimation. In particular, the table shows the number of events occurring in each of the time periods. This is useful information because the model is only estimable if each time period contains sufficient events.

Example 2 The piecewise constant exponential model contains the basic exponential model as a special case. Consequently, adding constraints to the parameters in the piecewise constant model should result in estimates for the basic exponential model. The `rate` command in Box 2 (command file `rt2c.cf`) illustrates this by adding 8 constraints in order to have the same baseline rate in all time periods. The result of this command should be identical to the output from `rt1.cf` shown in 6.17.2.1.

Box 1 Part of output from command file `rt2.cf`

```
Model: Exponential with time-periods

    Time period        Starting times  Ending times    Events
  ------------------------------------------------------------
   0.0000 -   12.0000             600           76        63
  12.0000 -   24.0000               0          104        96
  24.0000 -   36.0000               0          108        95
  36.0000 -   48.0000               0           55        44
  48.0000 -   60.0000               0           44        38
  60.0000 -   72.0000               0           38        29
  72.0000 -   84.0000               0           22        17
  84.0000 -   96.0000               0            9         6
  96.0000 -                         0          144        70

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Number of model parameters: 12
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1

Convergence reached in 5 iterations.
Number of function evaluations: 6 (6,6)

Idx SN Org Des MT Variable      Coeff       Error    C/Error  Signif
-------------------------------------------------------------------
  1  1   0   1  A Period-1    -5.2070      0.1520   -34.2464  1.0000
  2  1   0   1  A Period-2    -4.5473      0.1312   -34.6574  1.0000
  3  1   0   1  A Period-3    -4.2675      0.1296   -32.9389  1.0000
  4  1   0   1  A Period-4    -4.7784      0.1690   -28.2758  1.0000
  5  1   0   1  A Period-5    -4.7300      0.1779   -26.5924  1.0000
  6  1   0   1  A Period-6    -4.7862      0.1987   -24.0857  1.0000
  7  1   0   1  A Period-7    -5.1366      0.2523   -20.3568  1.0000
  8  1   0   1  A Period-8    -6.0783      0.4137   -14.6918  1.0000
  9  1   0   1  A Period-9    -5.4198      0.1281   -42.3123  1.0000
 10  1   0   1  A COHO2        0.4367      0.1133     3.8534  0.9999
 11  1   0   1  A COHO3        0.5162      0.1182     4.3688  1.0000
 12  1   0   1  A W            0.4336      0.0949     4.5688  1.0000

Log likelihood (starting values): -2514.0201
Log likelihood (final estimates): -2433.3757
```

Box 2 Illustration of constraints (`rt2c.cf`)

```
rate (
    tp = 0 (12) 96,
    xa(0,1) = COHO2,COHO3,W,

    con = b1 - b2 = 0,          # adding constraints
    con = b2 - b3 = 0,          # to request the same
    con = b3 - b4 = 0,          # baseline rate in
    con = b4 - b5 = 0,          # all periods
    con = b5 - b6 = 0,
    con = b6 - b7 = 0,
    con = b7 - b8 = 0,
    con = b8 - b9 = 0,

) = 3;
```

Relative Risks. Since the exponential model is a proportional transition rate model one can use the `rrisk` option as part of the `rate` command to request a table with relative risks. Calculation is done by simply taking the anti-logarithm of the estimated model parameters.

Generalized Residuals. If the model is specified for only a single transition, one can use the `pres` option to request generalized residuals. For each episode in the input data, the rate is calculated with formula (3), the (conditional) survivor function with formula (6), and the generalized residuals by taking the anti-logarithm of the survivor function.

For example, adding the `pres` option to the `rate` command in Box 2 would result in the same output as shown in **6.17.2.1** for the basic exponential model.

Printing Estimated Rates. The `prate` option can be used to request tables with estimated rates, survivor and density functions, depending on selected covariate constellations. For each point on the specified time axis, the rate is calculated according to (3) using the corresponding time period. Calculation of the survivor function is based on (6), always starting with time zero but integrating over all time periods up to the given time point. The density function is calculated by multiplying the rate and the survivor function.

6.17.2.3 Period-specific Effects

The piecewise constant exponential model described in 6.17.2.2 is based on the assumption that the effects of covariates do not change across time periods. Of course, the covariates may be time-dependent, but not their effects. An essentially different approach would allow also the effects of covariates to vary across time periods. This model with period-specific effects will be explained in the current section. We mention that this is different from a standard exponential model with interaction effects between covariates and time periods.

Implementation. The exponential model with time periods has model number 16; the command to request model estimation is

```
rate (tp=..., additional parameters) = 16;
```

The `tp` parameter must be used to provide time periods defined by split points on the time axis in the same way as described in 6.17.2.2 for the standard piecewise constant exponential model. Given these time periods, and using the same notation as in 6.17.2.2, the transition rate from origin state j to destination state k is

$$r_{jk}(t) = \exp\left\{\bar{\alpha}_l^{(jk)} + A^{(jk)}\alpha_l^{(jk)}\right\} \quad \text{if} \quad t \in I_l \tag{1}$$

For each transition (j,k), $\bar{\alpha}_l^{(jk)}$ is a constant coefficient associated with the lth time period. $A^{(jk)}$ is a (row) vector of covariates, and $\alpha_l^{(jk)}$ is an associated vector of coefficients, also associated with the lth time period. TDA estimates a null model, containing only parameters for the baseline rate, if specified without covariates. Covariates can be added to the A-term of the model with the option

```
xa(j,k) = list_of_variables,
```

The standard exponential model with time periods described in 6.17.2.2 is, of course, a special case of the model defined in (1). In fact, estimating the latter model with constraints so that the $\alpha_l^{(jk)}$ parameters are equal across time periods, would give the same result as a standard exponential model with time periods.

Box 1 Part of command file `rt3.cf`

```
rate (
    tp = 0 (12) 96,
    xa(0,1) = COHO2,COHO3,W,
) = 16;
```

Maximum Likelihood Estimation. The model is estimated following the outline given in 6.17.1.2 and 6.17.1.3. Using the same notation as in 6.17.2.2, the conditional survivor function can now be written as

$$G(t \mid s) = \exp\left\{-\sum_{l=1}^{m} \Delta[s,t,l]\exp(\bar{\alpha}_l + A\alpha_l)\right\} \tag{2}$$

Using this expression, the log-likelihood is

$$\ell = \sum_{i\in\mathcal{E}} (\bar{\alpha}_{l[t_i]} + A_i\alpha_{l[t_i]}) - \sum_{i\in\mathcal{N}}\sum_{l=1}^{m} \Delta[s_i,t_i,l]\exp(\bar{\alpha}_l + A_i\alpha_l) \tag{3}$$

and the first and second derivatives are

$$\begin{aligned}
\frac{\partial\ell}{\partial\bar{\alpha}_l} &= \sum_{i\in\mathcal{E}} \delta[t_i,l] + \sum_{i\in\mathcal{N}} -\Delta[s_i,t_i,l]\exp(\bar{\alpha}_l + A_i\alpha_l) \qquad (4)\\
\frac{\partial\ell}{\partial\alpha_{l,j}} &= \sum_{i\in\mathcal{E}} A_{i,j}\,\delta[t_i,l] + \sum_{i\in\mathcal{N}} -\Delta[s_i,t_i,l]\exp(\bar{\alpha}_l + A_i\alpha_l)\,A_{i,j}\\
\frac{\partial^2\ell}{\partial\bar{\alpha}_l\partial\bar{\alpha}_{l'}} &= \sum_{i\in\mathcal{N}} -\Delta[s_i,t_i,l]\exp(\bar{\alpha}_l + A_i\alpha_l)\,\delta_{ll'}\\
\frac{\partial^2\ell}{\partial\alpha_{l,j}\partial\alpha_{l',k}} &= \sum_{i\in\mathcal{N}}\sum_{l=1}^{m} -\Delta[s_i,t_i,l]\exp(\bar{\alpha}_l + A_i\alpha_l)\,A_{i,j}A_{i,k}\,\delta_{ll'}\\
\frac{\partial^2\ell}{\partial\bar{\alpha}_l\partial\alpha_{l',k}} &= \sum_{i\in\mathcal{N}} -\Delta[s_i,t_i,l]\exp(\bar{\alpha}_l + A_i\alpha_l)\,A_{i,k}\,\delta_{ll'}
\end{aligned}$$

Starting Values. Default starting values are calculated in the same way as for the piecewise constant exponential model described in 6.17.2.2, based on the results of a simple exponential null model.

Example 1 To illustrate estimation of an exponential model with period-specific effects we use again command file `rt1.cf`. The `rate` command

Box 2 Part of the output from command file `rt3.cf`

```
Model: Exponential with period-specific effects

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)
```

Idx	SN	Org	Des	MT	P	Variable	Coeff	Error	C/Error	Signif
1	1	0	1	A	1	Period-1	-5.0470	0.2696	-18.7220	1.0000
2	1	0	1	A	2	Period-2	-4.5147	0.2121	-21.2865	1.0000
3	1	0	1	A	3	Period-3	-4.1413	0.1949	-21.2431	1.0000
4	1	0	1	A	4	Period-4	-5.1107	0.3160	-16.1711	1.0000
5	1	0	1	A	5	Period-5	-4.9954	0.3226	-15.4869	1.0000
6	1	0	1	A	6	Period-6	-4.5366	0.2934	-15.4603	1.0000
7	1	0	1	A	7	Period-7	-5.5069	0.5127	-10.7403	1.0000
8	1	0	1	A	8	Period-8	-6.0808	0.7231	-8.4090	1.0000
9	1	0	1	A	9	Period-9	-5.5356	0.1732	-31.9616	1.0000
10	1	0	1	A	1	COHO2	0.4474	0.3362	1.3308	0.8168
11	1	0	1	A	2	COHO2	0.5783	0.2509	2.3050	0.9788
12	1	0	1	A	3	COHO2	0.4240	0.2411	1.7581	0.9213
13	1	0	1	A	4	COHO2	0.2850	0.3616	0.7881	0.5694
14	1	0	1	A	5	COHO2	0.6357	0.4089	1.5546	0.8800
15	1	0	1	A	6	COHO2	-0.0546	0.4292	-0.1273	0.1013
16	1	0	1	A	7	COHO2	1.5141	0.6137	2.4671	0.9864
17	1	0	1	A	8	COHO2	-0.8695	1.0971	-0.7925	0.5719
18	1	0	1	A	9	COHO2	0.2966	0.3020	0.9822	0.6740
19	1	0	1	A	1	COHO3	0.7676	0.3161	2.4284	0.9848
20	1	0	1	A	2	COHO3	0.5335	0.2609	2.0446	0.9591
21	1	0	1	A	3	COHO3	0.2488	0.2590	0.9606	0.6633
22	1	0	1	A	4	COHO3	0.3212	0.3684	0.8717	0.6166
23	1	0	1	A	5	COHO3	0.8420	0.3946	2.1337	0.9671
24	1	0	1	A	6	COHO3	-0.7163	0.5605	-1.2779	0.7987
25	1	0	1	A	7	COHO3	1.1975	0.6762	1.7708	0.9234
26	1	0	1	A	8	COHO3	-8.4412	42.1447	-0.2003	0.1587
27	1	0	1	A	9	COHO3	1.2489	0.3227	3.8701	0.9999
28	1	0	1	A	1	W	-0.1421	0.2620	-0.5422	0.4123
29	1	0	1	A	2	W	0.2518	0.2054	1.2259	0.7798
30	1	0	1	A	3	W	0.3492	0.2060	1.6958	0.9101
31	1	0	1	A	4	W	1.2122	0.3188	3.8026	0.9999
32	1	0	1	A	5	W	0.6285	0.3255	1.9310	0.9465
33	1	0	1	A	6	W	0.7757	0.3749	2.0691	0.9615
34	1	0	1	A	7	W	-0.2436	0.5366	-0.4539	0.3501
35	1	0	1	A	8	W	1.7909	0.8673	2.0650	0.9611
36	1	0	1	A	9	W	0.5607	0.2580	2.1729	0.9702

```
Log likelihood (starting values): -2514.0201
Log likelihood (final estimates): -2411.9909
```

is changed as shown in Box 1; the new command file is `rt3.cf`. For each covariate one gets m parameters, with m the number of time periods. So the total number of model parameters can easily become very large. To reduce the number of parameters one can use constraints. As already mentioned, constraining the parameters to get equal values across time periods would be equivalent to estimating the standard piecewise constant exponential model described in **6.17.2.2**. The example archive contains another command file, `rt3c.cf`, that illustrates how to use constraints.

Generalized Residuals. If the model is specified for only a single transition, one can use the `pres` option to request generalized residuals. For each episode in the input data, the rate is calculated with formula (1), the (conditional) survivor function with formula (2), and the generalized residuals by taking the anti-logarithm of the survivor function.

Printing Estimated Rates. The `prate` option is not supported for models with period-specific effects. This option might be added in a future version of TDA.

6.17.3 Parametric Duration Dependence

This section describes transition rate models with parametric duration dependence. All models can be estimated with the `rate` command, see 6.17.1.4. All examples use the episode data set `rrdat.1` that was discussed in 3.3.3.

6.17.3.1 Polynomial Rates

This section described two types of models where duration dependence is specified by a polynomial. In a direct approach, one can specify the transition rate by

$$r(t) = a + b_1 t + b_2 t^2 + \ldots b_n t^n \tag{1}$$

The corresponding survivor function is

$$G(t) = \exp\left\{-at - \frac{b_1}{2}t^2 - \frac{b_2}{3}t^3 - \ldots - \frac{b_n}{n+1}t^{n+1}\right\} \tag{2}$$

This model, sometimes called a generalized Rayleigh model, is described shortly by Lawless [1982, p. 252]. An obvious drawback is that not all possible parameter values will give sensible results.

With a small modification this problem can be avoided. If we take the approach

$$r(t) = \exp\left(a + b_1 t + b_2 t^2 + \ldots b_n t^n\right) \tag{3}$$

estimated rates can only take positive values. The corresponding conditional survivor function is

$$G(t \mid s) = \exp\left\{-\exp(a) \int_s^t \exp(b_1\tau + b_2\tau^2 + \ldots b_n\tau^n)\, d\tau\right\} \tag{4}$$

Unfortunately, there is no analytic solution; the integral must be evaluated numerically.

Implementation. Both types of models can be estimated with TDA. They are called models with polynomial rates, type I and II, respectively. The commands to request model estimation are

```
rate (parameter) = 4;
```

for the type I model, and

```
rate (parameter) = 5;
```

for the type II model. To specify the degree of the polynomial one can use the parameter

```
deg = degree_of_polynomial,
```

The default value is `deg=0`, both models are then identical to a simple exponential model. For type II model, one can also select a method for numerical integration with the `niset` command, see 5.4.1. Default is method 1 (`QNG` algorithm) with relative accuracy 0.0001.

1. The type I model provides for the possibility to link covariates to the parameter a via an exponential link function, the other parameters are estimated directly. The degree of the polynomial can be defined by the user, so there is the possibility of testing how many degrees are needed for a reasonably model fit. One gets the following formulation for the transition rate from origin state j to destination state k.

$$\begin{aligned} r_{jk}(t) &= a_{jk} + \sum_{l=1}^{n} b_l^{(jk)} t^l \\ a_{jk} &= \exp\left\{A^{(jk)}\alpha^{(jk)}\right\} \end{aligned} \tag{5}$$

2. In the type II model covariates can be linked to the parameter a by a linear link function, the other parameters are again estimated directly. Also the degree of the polynomial can be defined by the user. One then gets the following formulation for the transition rate from origin state j to destination state k.

$$\begin{aligned} r_{jk}(t) &= \exp\left\{a_{jk} + \sum_{l=1}^{n} b_l^{(jk)} t^l\right\} \\ a_{jk} &= A^{(jk)}\alpha^{(jk)} \end{aligned} \tag{6}$$

In both cases, $A^{(jk)}$ is a (row) vector of covariates, and $\alpha^{(jk)}$ is the associated vector of coefficients. The first component of $A^{(jk)}$ is supplied automatically by TDA as a constant equal to one. The model parameters to be estimated are the vectors

$$\alpha^{(jk)} \qquad \text{and} \qquad b^{(jk)} = (b_1^{(jk)}, \ldots, b_n^{(jk)})'$$

In both cases the null model is defined as a simple exponential model, that is a model with polynomial degree zero. We mention that the type I

model is not a proportional transition rate model, and the `rrisk` option is not available for this model.

Maximum Likelihood Estimation. The model is estimated following the outline given in **6.17.1.2** and **6.17.1.3**. To simplify notation we omit indices for transitions.

I. Using the conditional survivor function

$$G(t \mid s) = \exp\left\{a\,(s-t) + \sum_{l=1}^{n} \frac{b_l}{l+1}\,\left(s^{l+1} - t^{l+1}\right)\right\} \tag{7}$$

the log-likelihood for the type I model can be written as

$$\ell = \sum_{i\in\mathcal{E}} \log\left\{a + \sum_{l=1}^{n} b_l t_i^l\right\} + \sum_{i\in\mathcal{N}} \left\{a\,(s_i - t_i) + \sum_{l=1}^{n} \frac{b_l}{l+1}\,\left(s_i^{l+1} - t_i^{l+1}\right)\right\}$$

The first and second derivatives with respect to the α and b coefficients are

$$\frac{\partial \ell}{\partial \alpha_{j_a}} = \sum_{i\in\mathcal{E}} \frac{a}{a + \sum_l b_l t_i^l} A_{i,j_a} + \sum_{i\in\mathcal{N}} a\,(s_i - t_i)\,A_{i,j_a}$$

$$\frac{\partial \ell}{\partial b_{j_b}} = \sum_{i\in\mathcal{E}} \frac{1}{a + \sum_l b_l t_i^l}\, t_i^{j_b} + \sum_{i\in\mathcal{N}} \frac{\left(\frac{s_i}{t_i}\right)^{j_b}(s_i - t_i)}{j_b + 1}\, t_i^{j_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a}\partial \alpha_{k_a}} = \sum_{i\in\mathcal{E}} \frac{a}{a + \sum_l b_l t_i^l}\left\{1 - \frac{a}{a + \sum_l b_l t_i^l}\right\} A_{i,j_a} A_{i,k_a} + \sum_{i\in\mathcal{N}} a\,(s_i - t_i)\,A_{i,j_a} A_{i,k_a}$$

$$\frac{\partial^2 \ell}{\partial b_{j_b}\partial b_{k_b}} = \sum_{i\in\mathcal{E}} \frac{-1}{a + \sum_l b_l t_i^l}\, t_i^{j_b}\, t_i^{k_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a}\partial b_{k_b}} = \sum_{i\in\mathcal{E}} \frac{-a}{a + \sum_l b_l t_i^l}\, A_{i,j_a}\, t_i^{k_b}$$

II. For the type II model the integral needed in the formulation of (4) must be evaluated numerically. Assuming that a method for the evaluation of

$$\begin{aligned} I(s,t) &= \int_s^t \exp\Big\{\sum_{l=1}^n b_l \tau^l\Big\}\, d\tau \\ \frac{\partial I_n(s,t)}{\partial b_j} &= \int_s^t \exp\Big\{\sum_{l=1}^n b_l \tau^l\Big\}\, \tau^j\, d\tau \\ \frac{\partial^2 I_n(s_i,t_i)}{\partial b_j \partial b_k} &= \int_s^t \exp\Big\{\sum_{l=1}^n b_l \tau^l\Big\}\, \tau^{j+k}\, d\tau \end{aligned}$$

is available, the log-likelihood for the type II model can be written as

$$\ell = \sum_{i\in\mathcal{E}} \Big\{a + \sum_{l=1}^n b_l t_i^l\Big\} - \sum_{i\in\mathcal{N}} \exp(a)\, I(s_i,t_i)$$

and the first and second derivatives are

$$\begin{aligned} \frac{\partial \ell}{\partial \alpha_{j_a}} &= \sum_{i\in\mathcal{E}} A_{i,j_a} - \sum_{i\in\mathcal{N}} \exp(a)\, I(s_i,t_i)\, A_{i,j_a} \\ \frac{\partial \ell}{\partial b_{j_b}} &= \sum_{i\in\mathcal{E}} t_i^{j_b} - \sum_{i\in\mathcal{N}} \exp(a)\, \frac{\partial I(s_i,t_i)}{\partial b_{j_b}} \\ \frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \alpha_{k_a}} &= \sum_{i\in\mathcal{N}} -\exp(a)\, I(s_i,s_i)\, A_{i,j_a} A_{i,k_a} \\ \frac{\partial^2 \ell}{\partial b_{j_b} \partial b_{k_b}} &= \sum_{i\in\mathcal{N}} -\exp(a)\, \frac{\partial^2 I(s_i,t_i)}{\partial b_{j_b} \partial b_{k_b}} \\ \frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial b_{k_b}} &= \sum_{i\in\mathcal{N}} -\exp(a)\, \frac{\partial I(s_i,t_i)}{\partial b_{k_b}}\, A_{i,j_a} \end{aligned}$$

To evaluate the integrals, one can use the algorithms described in the chapter about numerical integration, see **5.4**.

Starting Values. Both types of the polynomial model contain the simple exponential model as a special case. Therefore it is possible to use the parameter estimates of correspondingly defined exponential null models

Box 1 Illustration of polynomial model, type I (`rt4.cf`)

```
Model: Polynomial (I) with degree 2

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Convergence reached in 7 iterations.
Number of function evaluations: 10 (10,10)

Maximum of log likelihood: -2454.04
Norm of final gradient vector: 0.00705957
Last absolute change of function value: 9.30148e-14
Last relative change in parameters: 2.72933e-06
Numerical problems: 1

Idx SN Org Des MT Variable        Coeff        Error      C/Error
-------------------------------------------------------------------
  1  1   0   1  A Constant  -4.4325e+00   1.0308e-01  -4.2999e+01
  2  1   0   1  A COHO2       2.4523e-01   8.9275e-02   2.7468e+00
  3  1   0   1  A COHO3       3.4108e-01   9.9069e-02   3.4429e+00
  4  1   0   1  A W           3.1746e-01   7.5806e-02   4.1878e+00
  5  1   0   1  B Beta-1     -7.6085e-05   1.5607e-05  -4.8750e+00
  6  1   0   1  B Beta-2      1.4274e-07   4.3506e-08   3.2809e+00

Log likelihood (starting values):  -2.5140e+03
Log likelihood (final estimates):  -2.4540e+03
```

as starting values. The default starting values used by TDA are calculated this way, i.e. the constant terms in the model parameters $\alpha^{(jk)}$ are initially set as in a simple exponential model, and all other model parameters are set to zero.

In most cases this is sufficient for low degree polynomials. If the degree of the polynomial is greater than three, the default starting values can result in convergence difficulties. In many cases this can be avoided if a series of models with successively higher degrees of the polynomial is estimated, and for each model the parameter estimates of the previous one are used as starting values.

Another source of convergence difficulties arises because of numerical overflow if the degree of the polynomials becomes large since TDA does not automatically scale variables of different magnitude.

Example 1 To illustrate the type I model, we use command file `rt4.cf` (not shown). Part of the standard output is shown in Box 1, the first line shows the `rate` command used for model estimation. In this example,

Box 2 Illustration of polynomial model, type II (`rt5.cf`)

```
Model: Polynomial (II) with degree 2
Numerical integration with Method 1 (QNG).
Relative error:  1.00000e-04

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Convergence reached in 7 iterations.
Number of function evaluations: 8 (8,8)

Maximum of log likelihood: -2454.54
Norm of final gradient vector: 5.24649e-07
Last absolute change of function value: 3.25666e-15
Last relative change in parameters: 3.81352e-06

Idx SN Org Des MT Variable        Coeff         Error       C/Error
-------------------------------------------------------------------
  1  1   0   1  A Constant  -4.5911e+00   1.1656e-01  -3.9389e+01
  2  1   0   1  A COHO2      3.8307e-01   1.1322e-01   3.3834e+00
  3  1   0   1  A COHO3      4.5832e-01   1.1780e-01   3.8908e+00
  4  1   0   1  A W          4.1726e-01   9.4840e-02   4.3996e+00
  5  1   0   1  B Beta-1    -3.6942e-03   2.1718e-03  -1.7010e+00
  6  1   0   1  B Beta-2    -7.4989e-06   9.6591e-06  -7.7635e-01

Log likelihood (starting values):  -2.5140e+03
Log likelihood (final estimates):  -2.4545e+03
```

we have selected a second-degree polynomial. Box 2 shows part of the output for an analogously specified type II model. The command file is `rt5.cf`.

Generalized Residuals. If one of the polynomial models is estimated for only a single transition, one can use the `pres` option to request a table with generalized residuals. For a type II model, the calculation uses again numerical integration.

Printing Estimated Rates. The `prate` option can be used to request tables with estimated rates, survivor and density functions, depending on selected covariate constellations.

6.17.3.2 Gompertz-Makeham Models

This section describes the Gompertz-Makeham model. The transition rate is given by the expression

$$r(t) = a + b\,\exp(c\,t) \qquad a, b \geq 0 \tag{1}$$

If $a = 0$, one gets the Gompertz model without a Makeham term. If $b = 0$, the model reduces to the simple exponential model. The corresponding survivor and density functions are

$$f(t) = \exp\left\{-at - \frac{b}{c}\left[\exp(c\,t) - 1\right]\right\}\left[a + b\,\exp(c\,t)\right] \tag{2}$$

$$G(t) = \exp\left\{-at - \frac{b}{c}\left[\exp(c\,t) - 1\right]\right\} \tag{3}$$

In the case of $c = 0$, it is assumed that these expressions reduce to the density and survivor function for a simple exponential model. Figure 1 shows some graphs of the transition rate function for $a = 0$, $b = 1$, and some different values of c.

Implementation. The Gompertz-Makeham model has three parameters to include covariates. TDA uses exponential link functions with the A- and B-term of the model, and a linear link function for the C-term. The model formulation for the transition rate from origin state j to destination state k is

$$\begin{aligned} r_{jk}(t) &= a_{jk} + b_{jk}\,\exp(c_{jk}\,t) \qquad (4)\\ a_{jk} &= \exp\left\{A^{(jk)}\alpha^{(jk)}\right\}\\ b_{jk} &= \exp\left\{B^{(jk)}\beta^{(jk)}\right\}\\ c_{jk} &= C^{(jk)}\gamma^{(jk)} \end{aligned}$$

It is assumed that the first component of each of the covariate (row) vectors $A^{(jk)}$, $B^{(jk)}$, and $C^{(jk)}$, is a constant equal to one. The associated coefficient vectors, $\alpha^{(jk)}$, $\beta^{(jk)}$, and $\gamma_{(jk)}$, are the model parameters to be estimated. Both, the Gompertz and the Gompertz-Makeham model

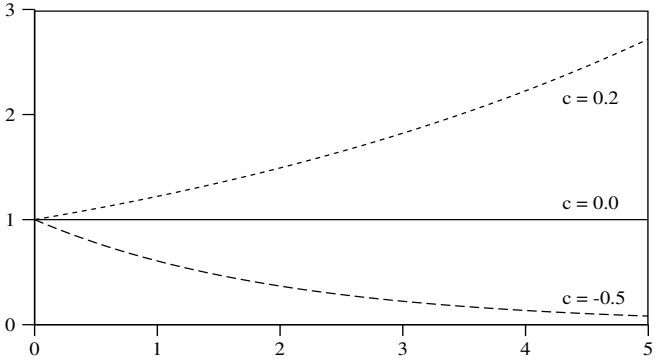


Figure 1 Gompertz transition rates. Plot created with command file `plot-gm.cf`.

have model number 6. If no covariates are linked to the A-term of the model, a simple Gompertz model is estimated. If at least one variable is linked to the A-term, the estimated model is of Gompertz-Makeham type. To estimate a Gompertz-Makeham model with the A-term a single constant, one can use the parameter `ni=1` to specify a single constant for the A term.

Maximum Likelihood Estimation. The model is estimated following the outline given in **6.17.1.2** and **6.17.1.3**. To simplify notation, we omit indices for transitions and use the abbreviations a, b, and c, as defined in (4). Providing for the possibility of episode splitting by using the conditional survivor function

$$G(t \mid s) = \exp\left\{a\,(s-t) + \frac{b}{c}\,[\exp(c\,s) - \exp(c\,t)]\right\} \tag{5}$$

one gets the log-likelihood

$$\ell = \sum_{i\in\mathcal{E}} \log\{a + b\exp(c\,t_i)\} + \sum_{i\in\mathcal{N}} a(s_i - t_i) + \frac{b}{c}\,[\exp(c\,s_i) - \exp(c\,t_i)\,]$$

The first and second derivatives with respect to the α, β, and γ coefficients are

$$\frac{\partial \ell}{\partial \alpha_{j_a}} = \sum_{i\in\mathcal{E}} \frac{a}{a + b\exp(c\,t_i)}\,A_{i,j_a} + \sum_{i\in\mathcal{N}} a\,(s_i - t_i)\,A_{i,j_a}$$

$$\frac{\partial \ell}{\partial \beta_{j_b}} = \sum_{i \in \mathcal{E}} \frac{b \exp(c\,t_i)}{a + b \exp(c\,t_i)} B_{i,j_b} + \sum_{i \in \mathcal{N}} \frac{b}{c} \left\{ \exp(c\,s_i) - \exp(c\,t_i) \right\}$$

$$\frac{\partial \ell}{\partial \gamma_{j_c}} = \sum_{i \in \mathcal{E}} \frac{b \exp(c\,t_i)}{a + b \exp(c\,t_i)} t_i\, C_{i,j_b} + \sum_{i \in \mathcal{N}} \frac{b}{c} \left\{ \exp(c\,s_i) \left[s_i - \frac{1}{c} \right] - \exp(c\,t_i) \left[t_i - \frac{1}{c} \right] \right\}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \alpha_{k_a}} = \sum_{i \in \mathcal{E}} \frac{a}{a + b \exp(c\,t_i)} \left\{ 1 - \frac{a}{a + b \exp(c\,t_i)} \right\} A_{i,j_a} A_{i,k_a} + \sum_{i \in \mathcal{N}} a\,(s_i - t_i)\, A_{i,j_a} A_{i,k_a}$$

$$\frac{\partial^2 \ell}{\partial \beta_{j_b} \partial \beta_{k_b}} = \sum_{i \in \mathcal{E}} \frac{b \exp(c\,t_i)}{a + b \exp(c\,t_i)} \left\{ 1 - \frac{b \exp(c\,t_i)}{a + b \exp(c\,t_i)} \right\} B_{i,j_b} B_{i,k_b} + \sum_{i \in \mathcal{N}} \frac{b}{c} \left\{ \exp(c\,s_i) - \exp(c\,t_i) \right\} B_{i,j_b} B_{i,k_b}$$

$$\frac{\partial^2 \ell}{\partial \gamma_{j_b} \partial \gamma_{k_b}} = \sum_{i \in \mathcal{E}} \frac{b \exp(c\,t_i)}{a + b \exp(c\,t_i)} \left\{ 1 - \frac{b \exp(c\,t_i)}{a + b \exp(c\,t_i)} \right\} t_i^2\, C_{i,j_b} C_{i,k_b} + \sum_{i \in \mathcal{N}} \frac{b}{c} \left\{ \exp(c\,s_i) \left[(s_i - \frac{1}{c})^2 + \frac{1}{c^2} \right] - \exp(c\,t_i) \left[(t_i - \frac{1}{c})^2 + \frac{1}{c^2} \right] \right\} C_{i,j_b} C_{i,k_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \beta_{k_b}} = \sum_{i \in \mathcal{E}} \frac{-a\,b \exp(c\,t_i)}{\left[a + b \exp(c\,t_i) \right]^2} A_{i,j_a} B_{i,k_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \gamma_{k_b}} = \sum_{i \in \mathcal{E}} \frac{-a\,b \exp(c\,t_i)}{\left[a + b \exp(c\,t_i) \right]^2} t_i\, A_{i,j_a} C_{i,k_b}$$

$$\frac{\partial^2 \ell}{\partial \beta_{j_a} \partial \gamma_{k_b}} = \sum_{i \in \mathcal{E}} \frac{b \exp(c\,t_i)}{a + b \exp(c\,t_i)} \left\{ 1 - \frac{b \exp(c\,t_i)}{a + b \exp(c\,t_i)} \right\} t_i\, B_{i,j_b} C_{i,k_c} + \sum_{i \in \mathcal{N}} \frac{b}{c} \left\{ \exp(c\,s_i) \left[s_i - \frac{1}{c} \right] - \exp(c\,t_i) \left[t_i - \frac{1}{c} \right] \right\} B_{i,j_b} C_{i,k_c}$$

Box 1 Output from command file `rt6.cf`

```
Model: Gompertz-Makeham

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Number of model parameters: 8
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Log-likelihood of exponential null model: -2514.02
Scaling factor for log-likelihood: -0.001
Using default starting values.

Convergence reached in 6 iterations.
Number of function evaluations: 8 (8,8)

Maximum of log likelihood: -2453.38
Norm of final gradient vector: 1.83153e-06
Last absolute change of function value: 2.81327e-12
Last relative change in parameters: 8.91402e-05

Idx SN Org Des MT Variable      Coeff      Error    C/Error  Signif
-----------------------------------------------------------------------
  1  1   0   1  B Constant    -4.5114     0.1198   -37.6503  1.0000
  2  1   0   1  B COHO2        0.4595     0.1546     2.9716  0.9970
  3  1   0   1  B COHO3        0.4147     0.1657     2.5025  0.9877
  4  1   0   1  B W            0.2860     0.1280     2.2351  0.9746
  5  1   0   1  C Constant    -0.0060     0.0013    -4.6239  1.0000
  6  1   0   1  C COHO2       -0.0016     0.0022    -0.7275  0.5331
  7  1   0   1  C COHO3        0.0011     0.0028     0.3986  0.3098
  8  1   0   1  C W            0.0026     0.0018     1.4299  0.8472

Log likelihood (starting values): -2557.3645
Log likelihood (final estimates): -2453.3828
```

Starting Values. In most cases, good initial estimates are required to estimate Gompertz-Makeham models. It is difficult, however, to determine such estimates since it depends on whether a positive or negative time dependence should be assumed. In particular the addition of a Gompertz A-term in the model specification can give rise to conver-

gence difficulties. Default starting values used with TDA are calculated by using the constant rate of a correspondingly defined exponential null model. The constant coefficients in the A- and the B-terms of the model are taken to be the logarithm of this rate, the constant coefficient in the C-term is taken to be minus this rate. The idea is to assume negative time dependence as the "normal case", and to start with some estimate of the mean duration. However, the starting values calculated this way are not very good initial estimates. So it may be necessary to try other ones.

Example 1 As is seen from Figure 1, the Gompertz-Makeham model is not appropriate for our main example data. As a consequence we find severe convergence difficulties when trying a full Gompertz-Makeham specification. No problems arise, however, if we try a model without the Makeham term. To illustrate, we use command file `rt6.cf`, identical with `rt1.cf` except for the model number that has been changed into 6. Box 1 shows the resulting parameter estimates.

Generalized Residuals. For the Gompertz-Makeham models, one can use the `pres` option to request a table with generalized residuals.

Printing Estimated Rates. The `prate` option can be used to request tables with estimated rates, survivor and density functions, depending on selected covariate constellations.

6.17.3.3 Weibull Models

This section describes the Weibull model. In the single transition case it is derived by assuming the duration variable T to follow a Weibull distribution. Density and survivor function, and the transition rate, are given respectively by

$$f(t) = b\,a^b\,t^{b-1}\exp\left\{-(at)^b\right\} \qquad a,b>0 \tag{1}$$

$$G(t) = \exp\left\{-(at)^b\right\} \tag{2}$$

$$r(t) = b\,a^b\,t^{b-1} \tag{3}$$

Figure 1 shows graphs of the transition rate for $a = 1$ and some different values of b.

Implementation. The Weibull model has two parameters, so one has two possibilities to include covariates. Using standard exponential link functions, the model formulation for the transition rate from origin state j to destination state k is

$$r_{jk}(t) = b_{jk}\,a_{jk}^{b_{jk}}\,t^{b_{jk}-1} \tag{4}$$

$$a_{jk} = \exp\left\{A^{(jk)}\alpha^{(jk)}\right\}$$

$$b_{jk} = \exp\left\{B^{(jk)}\beta^{(jk)}\right\}$$

It is assumed that the first component of each of the covariate (row) vectors $A^{(jk)}$ and $B^{(jk)}$ is a constant equal to one. The associated coefficient vectors, $\alpha^{(jk)}$ and $\beta^{(jk)}$, are the model parameters to be estimated. The Weibull model has model number 7.

Maximum Likelihood Estimation. The model is estimated following the outline given in **6.17.1.2** and **6.17.1.3**. To simplify the notation we omit indices for transitions and use the abbreviations a and b as defined in (4). Using the conditional survivor function

$$G(t \mid s) = \frac{\exp(-(at)^b)}{\exp(-(as)^b)} \tag{5}$$

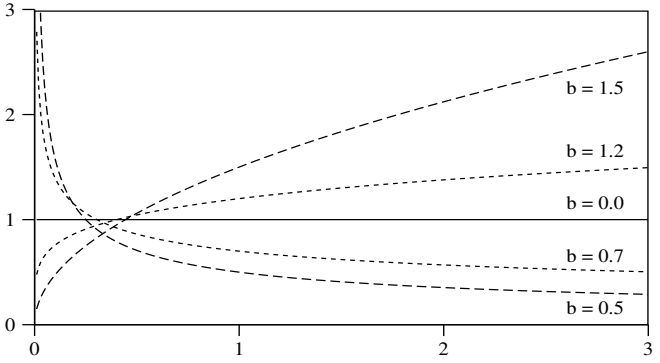


Figure 1 Weibull transition rates. Plot created with command file `plot-wei.cf`.

the log-likelihood is

$$\ell = \sum_{i\in\mathcal{E}} \log\left(b\, a^b\, t_i^{b-1}\right) + \sum_{i\in\mathcal{N}} (as_i)^b - (at_i)^b \tag{6}$$

and the first and second derivatives with respect to the α and β coefficients are

$$\frac{\partial \ell}{\partial \alpha_{j_a}} = \sum_{i\in\mathcal{E}} b\, A_{i,j_a} + \sum_{i\in\mathcal{N}} b\left\{(as_i)^b - (at_i)^b\right\} A_{i,j_a}$$

$$\frac{\partial \ell}{\partial \beta_{j_b}} = \sum_{i\in\mathcal{E}} \{1 + b\,\log(at_i)\}\, B_{i,j_b} + \sum_{i\in\mathcal{N}} b\left\{\log(as_i)(as_i)^b - \log(at_i)(at_i)^b\right\} B_{i,j_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a}\partial \alpha_{k_a}} = \sum_{i\in\mathcal{N}} b^2\left\{(as_i)^b - (at_i)^b\right\} A_{i,j_a} A_{i,k_a}$$

$$\frac{\partial^2 \ell}{\partial \beta_{j_b}\partial \beta_{k_b}} = \sum_{i\in\mathcal{E}} b\,\log(at_i) B_{i,j_b} B_{i,k_b} + \sum_{i\in\mathcal{N}} b\left\{\log(as_i)(as_i)^b\,[1 + b\log(as_i)] - \log(at_i)(at_i)^b\,[1 + b\log(at_i)]\right\} B_{i,j_b} B_{i,k_b}$$

Box 1 Output from command file `rt7.cf`

```
Model: Weibull

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Number of model parameters: 5
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Convergence reached in 6 iterations.
Number of function evaluations: 7 (7,7)

Maximum of log likelihood: -2473.54
Norm of final gradient vector: 4.34528e-08
Last absolute change of function value: 5.00438e-12
Last relative change in parameters: 4.56362e-05

Idx SN Org Des MT Variable      Coeff       Error    C/Error  Signif
-------------------------------------------------------------------
  1  1   0   1  A Constant    -5.0055      0.0906   -55.2452  1.0000
  2  1   0   1  A COHO2        0.5339      0.1201     4.4448  1.0000
  3  1   0   1  A COHO3        0.6671      0.1235     5.4003  1.0000
  4  1   0   1  A W            0.5191      0.1012     5.1274  1.0000
  5  1   0   1  B Constant    -0.0700      0.0366    -1.9119  0.9441

Log likelihood (starting values): -2514.0201
Log likelihood (final estimates): -2473.5432
```

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \beta_{k_b}} = \sum_{i \in \mathcal{E}} b\, A_{i,j_a} B_{i,k_b} + \sum_{i \in \mathcal{N}} b \left\{ (as_i)^b \left[1 + b \log(as_i)\right] - (at_i)^b \left[1 + b \log(at_i)\right] \right\} A_{i,j_a} B_{i,k_b}$$

Starting Values. Estimation of Weibull models is relatively robust with respect to starting values. Therefore initial estimates used by TDA are taken to be the constant rates of an accordingly defined exponential

null model. All model coefficients, without the constant coefficient of the A-term, are set to zero, implying a unit shape parameter of the duration distribution, i.e. an exponential model.

Example 1 Although the Weibull model does not seem appropriate for our main example data, just to illustrate its estimation we use command file `rt7.cf`. Estimation results are shown in Box 1.

Generalized Residuals. For the Weibull model, one can use the `pres` option to request a table with generalized residuals.

Printing Estimated Rates. The `prate` option can be used to request tables with estimated rates, survivor and density functions, depending on selected covariate constellations. Note that, for $t = 0$, the rate and density is always set to zero, and the survivor function is set to 1; this might not be correct.

6.17.3.4 Sickle Models

This section describes the Sickle model proposed by Diekmann and Mitter ([1983], [1984]). Formulated first for the single transition case, it starts with the assumption of a transition rate given by the expression

$$r(t) = a\,t\,\exp\left\{-\frac{t}{b}\right\} \qquad a, b > 0 \tag{1}$$

The corresponding survivor and density functions are

$$f(t) = \exp\left\{-a\,b\,[\,b - (t+b)\exp(-\frac{t}{b})]\right\}\,a\,t\exp\left\{-\frac{t}{b}\right\} \tag{2}$$

$$G(t) = \exp\left\{-a\,b\,[\,b - (t+b)\exp(-\frac{t}{b})]\right\} \tag{3}$$

As is easily seen, this distribution is defective. The survivor function goes not to zero if $t \to \infty$, instead one finds that

$$\lim_{t\to\infty} G(t) = \exp(-a\,b^2) \tag{4}$$

Figure 1 shows graphs of the rate function for $a = 1$ and some values of b. It looks like a sickle, and this has given the name for the model.

Implementation. The Sickle model has two parameters, so there are two possibilities to include covariates. Because both of the parameters must be positive, TDA uses standard exponential link functions. One then gets the following model formulation for the transition rate from origin state j to destination state k.

$$r_{jk}(t) = a_{jk}\,t\,\exp\left\{-\frac{t}{b_{jk}}\right\} \tag{5}$$

$$a_{jk} = \exp\left\{A^{(jk)}\alpha^{(jk)}\right\}$$

$$b_{jk} = \exp\left\{B^{(jk)}\beta^{(jk)}\right\}$$

It is assumed that the first component of each of the covariate (row) vectors $A^{(jk)}$ and $B^{(jk)}$ is a constant equal to one. The associated coefficient vectors, $\alpha^{(jk)}$ and $\beta^{(jk)}$, are the model parameters to be estimated. The sickle model has model number 8.

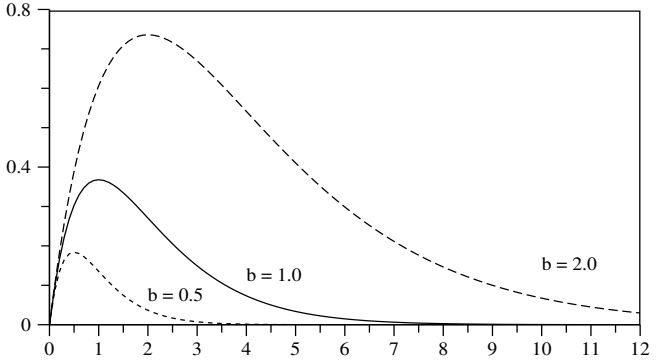


Figure 1 Sickle transition rates. Plot created with command file `plot-sic.cf`.

Maximum Likelihood Estimation. The model is estimated following the outline given in **6.17.1.2** and **6.17.1.3**. To simplify the notation we omit indices for transitions and use the abbreviations a and b as defined in (5). Providing for the possibility of episode splitting by using the conditional survivor function

$$G(t \mid s) = \exp\left\{ a\, b \left[(t+b)\exp(-\frac{t}{b}) - (s+b)\exp\left(-\frac{s}{b}\right) \right] \right\}$$

the log-likelihood is

$$\ell = \sum_{i \in \mathcal{E}} \log(a\, t_i) - \frac{t_i}{b} + \sum_{i \in \mathcal{N}} a\, b \left\{ (t_i + b)\exp\left(-\frac{t_i}{b}\right) - (s_i + b)\exp\left(-\frac{s_i}{b}\right) \right\}$$

and the first and second derivatives with respect to the α and β coefficients are

$$\frac{\partial \ell}{\partial \alpha_{j_a}} = \sum_{i \in \mathcal{E}} A_{i,j_a} + \sum_{i \in \mathcal{N}} a\, b \left\{ (t_i + b)\exp(-\frac{t_i}{b}) - (s_i + b)\exp(-\frac{s_i}{b}) \right\} A_{i,j_a}$$

$$\frac{\partial \ell}{\partial \beta_{j_b}} = \sum_{i\in\mathcal{E}} \frac{t_i}{b} B_{i,j_b} + \sum_{i\in\mathcal{N}} a\,b \left\{ \left[2(t_i+b) + \frac{t_i^2}{b} \right] \exp(-\frac{t_i}{b}) - \left[2(s_i+b) + \frac{s_i^2}{b} \right] \exp(-\frac{s_i}{b}) \right\} B_{i,j_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \alpha_{k_a}} = \sum_{i\in\mathcal{N}} a\,b \left\{ (t_i+b)\exp(-\frac{t_i}{b}) - (s_i+b)\exp(-\frac{s_i}{b}) \right\} A_{i,j_a} A_{i,k_a}$$

$$\frac{\partial^2 \ell}{\partial \beta_{j_b} \partial \beta_{k_b}} = \sum_{i\in\mathcal{E}} -\frac{t_i}{b} B_{i,j_b} B_{i,k_b} + \sum_{i\in\mathcal{N}} a\,b \left\{ \left[4(t_i+b) + (2+\frac{t_i}{b})\frac{t_i^2}{b} \right] \exp(-\frac{t_i}{b}) - \left[4(s_i+b) + (2+\frac{s_i}{b})\frac{s_i^2}{b} \right] \exp(-\frac{s_i}{b}) \right\} B_{i,j_b} B_{i,k_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \beta_{k_b}} = \sum_{i\in\mathcal{N}} a\,b \left\{ \left[2(t_i+b) + \frac{t_i^2}{b} \right] \exp(-\frac{t_i}{b}) - \left[2(s_i+b) + \frac{s_i^2}{b} \right] \exp(-\frac{s_i}{b}) \right\} A_{i,j_a} B_{i,k_b}$$

Starting Values. The choice of starting values for the Sickle model seems not to be critical. In most applications it seems to be sufficient to set initially all coefficients to zero. Somewhat more efficient calculations are possible if one chooses as initial estimates $\alpha_0 = r^2$ and $\beta_0 = -\log(r)$ with r the constant rate of a correspondingly defined exponential null model. The starting values used by TDA are defined this way.

Example 1 To illustrate estimation of the Sickle model, we use our main example data; the command file is `rt8.cf`, identical with `rt1.cf`, only the model number has been changed into 8. Estimation results are shown in Box 1. Note that the program has given an information about numerical problems. Number 4 signals that the Hessian matrix was at least for one time not positive definite during the iterations to maximize the likelihood. However, in the final iteration steps it was positive definite, and so there is no real problem. For more details about the iteration process one can request a protocol file with the `pprot` parameter.

Box 1 Output from command file `rt8.cf`

```
Model: Sickle

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Number of model parameters: 5
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Log-likelihood of exponential null model: -2514.02
Scaling factor for log-likelihood: -0.001
Using default starting values.

Convergence reached in 7 iterations.
Number of function evaluations: 10 (10,10)

Maximum of log likelihood: -2466.1
Norm of final gradient vector: 4.84093e-06
Last absolute change of function value: 1.11023e-09
Last relative change in parameters: 0.00044869

Numerical problems.
  4 : Hessian not positive definite.

Idx SN Org Des MT Variable      Coeff      Error    C/Error  Signif
-------------------------------------------------------------------
  1  1   0   1  A Constant    -7.0645     0.1140   -61.9618  1.0000
  2  1   0   1  A COHO2         0.3926     0.1123     3.4947  0.9995
  3  1   0   1  A COHO3         0.4363     0.1162     3.7556  0.9998
  4  1   0   1  A W             0.4232     0.0944     4.4808  1.0000
  5  1   0   1  B Constant      3.7046     0.0517    71.6511  1.0000

Log likelihood (starting values): -2897.7621
Log likelihood (final estimates): -2466.1005
```

Generalized Residuals. For the sickle model, one can use the `pres` option to request a table with generalized residuals.

Printing Estimated Rates. The `prate` option can be used to request tables with estimated rates, survivor and density functions, depending on selected covariate constellations.

6.17.3.5 Log-logistic Models

Another model with time-varying transition rates is the log-logistic. This section describes the standard log-logistic model and an extension.

I. In the single transition case the standard log-logistic model is based on the assumption that the duration variable follows a log-logistic distribution. The density, survivor, and rate functions for this distribution are

$$f(t) = \frac{b\,a^b\,t^{b-1}}{[1+(at)^b]^2} \qquad a,b>0 \tag{1}$$

$$G(t) = \frac{1}{1+(at)^b} \tag{2}$$

$$r(t) = \frac{b\,a^b\,t^{b-1}}{1+(at)^b} \tag{3}$$

Figure 1 shows graphs of the rate function for $a=1$ and some values of b. The time t_{max} when the rate reaches its maximum, $r_{\max}$, is given by

$$t_{\max} = \frac{1}{a}\,(b-1)^{\frac{1}{b}} \qquad r_{\max} = a\,(b-1)^{1-\frac{1}{b}} \tag{4}$$

II. A sometimes useful extension of the standard log-logistic model was proposed by Brüderl [1991]; see also Brüderl and Diekmann [1995]. The idea is to include a third parameter to allow for varying levels of the transition rate. The model definition is

$$r(t) = c\,\frac{b\,(a\,t)^{b-1}}{1+(at)^b} \qquad a,b,c>0 \tag{5}$$

The associated density and survivor functions are

$$f(t) = c\,\frac{b\,(a\,t)^{b-1}}{[1+(at)^b]^{\frac{c}{a}+1}}$$

$$G(t) = \frac{1}{[1+(at)^b]^{\frac{c}{a}}}$$

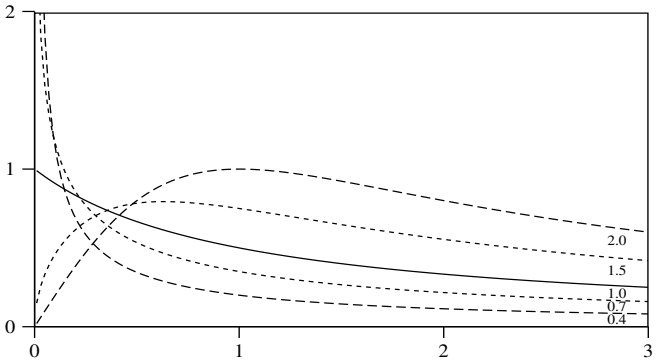


Figure 1 Log-logistic transition rates. Plot created with command file `plot-ll.cf`.

Implementation. Both versions of the log-logistic transition rate model can be estimated, the model numbers are 9 and 10, respectively.

I. The standard log-logistic model has two parameters, so there are two possibilities to include covariates. TDA uses exponential link functions, so one gets the following model formulation for the transition rate from origin state j to destination state k.

$$r_{jk}(t) = \frac{b_{jk}\, a_{jk}^{b_{jk}}\, t^{b_{jk}-1}}{1 + (a_{jk}t)^{\,b_{jk}}}$$

$$a_{jk} = \exp\left\{A^{(jk)}\alpha^{(jk)}\right\}$$

$$b_{jk} = \exp\left\{B^{(jk)}\beta^{(jk)}\right\}$$

II. The extended log-logistic model has three parameters, providing three possibilities to include covariates. Using again standard exponential link functions, one gets the following model formulation for the transition rate from origin state j to destination state k.

$$r_{jk}(t) = c_{jk}\, \frac{b_{jk}\,(a_{jk}\,t)^{b_{jk}-1}}{1 + (a_{jk}t)^{b_{jk}}}$$

$$a_{jk} = \exp\left\{A^{(jk)}\alpha^{(jk)}\right\}$$

$$b_{jk} = \exp\left\{B^{(jk)}\beta^{(jk)}\right\}$$

$$c_{jk} = \exp\left\{C^{(jk)}\gamma^{(jk)}\right\}$$

Again it is assumed that the first component of each of the covariate vectors $A^{(jk)}$, $B^{(jk)}$, and $C^{(jk)}$, is a constant equal to one. The coefficients $\alpha^{(jk)}$, $\beta^{(jk)}$, and $\gamma^{(jk)}$, are the model parameters to be estimated.

Maximum Likelihood Estimation. The model is estimated following the outline given in **6.17.1.2** and **6.17.1.3**. To simplify the notation we omit indices for transitions and use some already defined abbreviations.

I. The conditional survivor function for the standard log-logistic model may be written as

$$G(t \mid s) = \frac{1 + (as)^b}{1 + (at)^b} \tag{6}$$

Using these conditional survivor functions, the log-likelihood is

$$\ell = \sum_{i \in \mathcal{E}} \log\left\{\frac{b\, a^b\, t_i^{b-1}}{1 + (at_i)^b}\right\} + \sum_{i \in \mathcal{N}} \log\left\{\frac{1 + (as_i)^b}{1 + (at_i)^b}\right\} \tag{7}$$

with first and second derivatives

$$\frac{\partial \ell}{\partial \alpha_{j_a}} = \sum_{i \in \mathcal{E}} b\left\{1 - \frac{(at_i)^b}{1 + (at_i)^b}\right\} A_{i,j_a} + \sum_{i \in \mathcal{N}} b\left\{\frac{(as_i)^b}{1 + (as_i)^b} - \frac{(at_i)^b}{1 + (at_i)^b}\right\} A_{i,j_a}$$

$$\frac{\partial \ell}{\partial \beta_{j_b}} = \sum_{i \in \mathcal{E}} \left\{1 + b\,\frac{\log(at_i)}{1 + (at_i)^b}\right\} B_{i,j_b} + \sum_{i \in \mathcal{N}} b\left\{\log(as_i)\frac{(as_i)^b}{1 + (as_i)^b} - \log(at_i)\frac{(at_i)^b}{1 + (at_i)^b}\right\} B_{i,j_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \alpha_{k_a}} = \sum_{i \in \mathcal{E}} b^2 \frac{(at_i)^b}{1+(at_i)^b} \left\{ \frac{(at_i)^b}{1+(at_i)^b} - 1 \right\} A_{i,j_a} A_{i,k_a} + \sum_{i \in \mathcal{N}} b^2 \left\{ \frac{(as_i)^b}{1+(as_i)^b} \left[1 - \frac{(as_i)^b}{1+(as_i)^b} \right] - \frac{(at_i)^b}{1+(at_i)^b} \left[1 - \frac{(at_i)^b}{1+(at_i)^b} \right] \right\} A_{i,j_a} A_{i,k_a}$$

$$\frac{\partial^2 \ell}{\partial \beta_{j_b} \partial \beta_{k_b}} = \sum_{i \in \mathcal{E}} b \frac{\log(at_i)}{1+(at_i)^b} \left\{ 1 - b \log(at_i) \frac{(at_i)^b}{1+(at_i)^b} \right\} B_{i,j_b} B_{i,k_b} + \sum_{i \in \mathcal{N}} b \left\{ \log(as_i) \frac{(as_i)^b}{1+(as_i)^b} \left[1 + b \frac{\log(as_i)}{1+(as_i)^b} \right] - \log(at_i) \frac{(at_i)^b}{1+(at_i)^b} \left[1 + b \frac{\log(at_i)}{1+(at_i)^b} \right] \right\} B_{i,j_b} B_{i,k_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \beta_{k_b}} = \sum_{i \in \mathcal{E}} b \left\{ 1 - \frac{(at_i)^b}{1+(at_i)^b} \left[1 + \frac{b\, log(at_i)}{1+(at_i)^b} \right] \right\} A_{i,j_a} B_{i,k_b} + \sum_{i \in \mathcal{N}} b \left\{ \frac{(as_i)^b}{1+(as_i)^b} \left[1 + \frac{b\, log(as_i)}{1+(as_i)^b} \right] - \frac{(at_i)^b}{1+(at_i)^b} \left[1 + \frac{b\, log(at_i)}{1+(at_i)^b} \right] \right\} A_{i,j_a} B_{i,k_b}$$

Starting Values. Starting values for estimating the standard log-logistic model are not critical, and in most situations it is sufficient to set the shape parameter $b = 1$ and to use for the parameter a the constant rate of an accordingly defined exponential null model. TDA uses this method to calculated default starting values.

For the extended models the choice of starting values is critical. In many cases the best way to reach convergence is to use as starting values the parameter estimates of a standard log-logistic model, and to set the C-term coefficients to zero. Because of the overhead involved in this method it is not practical as a default. Default starting values are therefore calculated by TDA in another way: the A- and B-term coefficients of the model are set to zero, and the C-term coefficients are defined by the constant rate of an accordingly defined exponential null model. For many situations this method seems to provide sufficient initial parameter estimates.

Box 1 Output from command file `rt9.cf`

```
Model: Log-logistic (I)

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Number of model parameters: 5
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Log-likelihood of exponential null model: -2514.02
Scaling factor for log-likelihood: -0.001
Using default starting values.

Convergence reached in 5 iterations.
Number of function evaluations: 8 (8,8)

Maximum of log likelihood: -2442.3
Norm of final gradient vector: 3.28158e-05
Last absolute change of function value: 6.67562e-09
Last relative change in parameters: 0.000446382

Idx SN Org Des MT Variable      Coeff      Error    C/Error  Signif
-------------------------------------------------------------------
  1  1   0   1  A Constant    -4.3546     0.0998   -43.6183  1.0000
  2  1   0   1  A COHO2        0.4792     0.1257     3.8138  0.9999
  3  1   0   1  A COHO3        0.5927     0.1295     4.5752  1.0000
  4  1   0   1  A W            0.3989     0.1065     3.7463  0.9998
  5  1   0   1  B Constant     0.3307     0.0385     8.5823  1.0000

Log likelihood (starting values): -2521.7850
Log likelihood (final estimates): -2442.3017
```

Example 1 To illustrate estimation of the standard (type I) log-logistic model, we use command file `rt9.cf`. Parameter estimates are shown in Box 1.

Generalized Residuals. For both types of the log-logistic model, one can use the `pres` option to request a table with generalized residuals.

Printing Estimated Rates. The `prate` option can be used to request tables with estimated rates, survivor and density functions, depending on selected covariate constellations. Tor $t = 0$, the rate and density is always set to zero, and the survivor function is set to 1; this might not be correct. The `rrisk` option cannot be used with the standard log-logistic models. For the extended (type II) log-logistic model, the `rrisk` option shows relative risks for covariates linked to the C-term of the model.

6.17.3.6 Log-normal Models

This section describes two versions of the log-normal model, a standard log-normal model and a model with an additional shift-parameter. The models correspond to the two-parameter and the three-parameter log-normal distributions as described, for instance, by Aitchison and Brown [1957]. (Descriptions of log-normal rate models are given by Lawless [1982], p. 313, Lancaster [1990], p. 47.)

In the single transition case, the standard (two-parameter) log-normal model is derived by assuming that the logarithm of the duration variable, T, follows a normal distribution or, equivalently, that T follows a log-normal distribution with density function

$$f(t) = \frac{1}{b\,t}\,\phi\left(\frac{\log(t) - a}{b}\right) \qquad b > 0 \tag{1}$$

ϕ and Φ are used, respectively, to denote the standard normal density and distribution function:

$$\phi(\tau) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{\tau^2}{2})$$

$$\Phi(t) = \int_0^t \phi\,(\tau)\,d\,\tau$$

The survivor function corresponding to (1) is

$$G(t) = 1 - \Phi\left(\frac{\log(t) - a}{b}\right) \tag{2}$$

and the transition rate can be written as

$$r(t) = \frac{1}{b\,t}\,\frac{\phi\,(z_t)}{1 - \Phi\,(z_t)} \qquad \text{where} \quad z_t = \frac{\log(t) - a}{b}$$

Figure 1 shows graphs of the rate function for $a = 0$ and some different values of b. As is seen, the graphs are very similar for the log-normal and the log-logistic model, provided that $b > 1$ in the latter case.

A simple, sometimes useful extension of the standard log-normal model is derived by adding a shift parameter $c > 0$. The assumption,

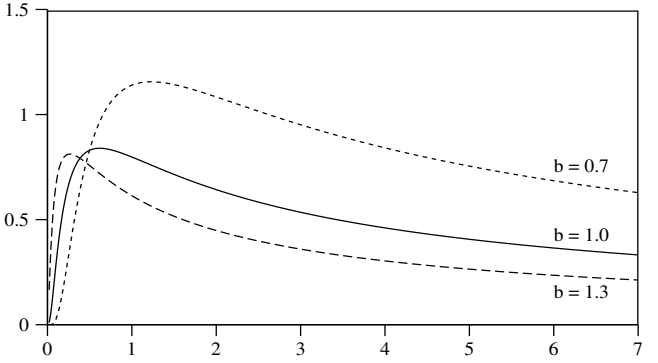


Figure 1 Log-normal transition rates. Plot created with command file `plot-ln.cf`.

then, is that not T, but $T-c$ follows a log-normal distribution. Of course, this assumption implies that no episode ends before c, meaning that c can be thought of as a basic waiting time until the risk of leaving the origin state becomes positive. The resulting model formulation is very similar to the standard log-normal model. The density, survivor and rate functions have the same expressions as given above for the standard case, one only has to substitute t by $t-c$. The result is a simple shift of the functions to the right by an amount of c.

Implementation. The standard log-normal model has two parameters, so there are two possibilities to include covariates. Following Lawless ([1982], p. 313), TDA uses a linear link function for a, but additionally provides the possibility to link covariates to the dispersion parameter, b, via an exponential link function. So one gets the following model formulation for the transition rate from origin state j to destination state k.

$$r_{jk}(t) = \frac{1}{b_{jk}\, t} \frac{\phi\left(z_t^{(jk)}\right)}{1-\Phi\left(z_t^{(jk)}\right)}$$

$$z_t^{(jk)} = \frac{\log(t)-a_{jk}}{b_{jk}}$$

$$a_{jk} = A^{(jk)}\alpha^{(jk)}$$

$$b_{jk} = \exp\left\{B^{(jk)}\beta^{(jk)}\right\}$$

The extended three-parameter model is defined analogously by

$$r_{jk}(t) = \frac{1}{b_{jk}\,(t - c_{jk})}\,\frac{\phi\left(z_t^{(jk)}\right)}{1 - \Phi\left(z_t^{(jk)}\right)} \tag{3}$$

$$z_t^{(jk)} = \frac{\log(t - c_{jk}) - a_{jk}}{b_{jk}}$$

$$a_{jk} = A^{(jk)}\alpha^{(jk)}$$

$$b_{jk} = \exp\left\{B^{(jk)}\beta^{(jk)}\right\}$$

$$c_{jk} = \exp\left\{C^{(jk)}\gamma^{(jk)}\right\}$$

In both cases, it is assumed that the first component of each of the covariate (row) vectors $A^{(jk)}$, $B^{(jk)}$, and $C^{(jk)}$ in the extended model, is a constant equal to one. The associated coefficient vectors $\alpha^{(jk)}$, $\beta^{(jk)}$, and $\gamma^{(jk)}$, are the model parameters to be estimated.

Both versions of the log-normal model have the same model numbers, 12. The two versions are distinguished by the definition of a C-term of the model. If this term is defined, TDA estimates an extended model, otherwise a standard log-normal model. To request estimation of a log-normal model of type II with a single constant in the C-term, one can use the parameter `ni=1` to link a single constant to the C term.

Maximum Likelihood Estimation. The model is estimated following the outline given in **6.17.1.2** and **6.17.1.3**. Since the standard model is a special case of the extended model, when $c = 0$, we only give formulas for the extended model. To simplify the notation we omit indices for transitions and use the abbreviations a, b, c, and z_t, as defined in (3). Furthermore, we define

$$Q(z) = 1 - \Phi(z) \quad \text{and} \quad V(z) = \frac{\phi(z)}{Q(z)} \tag{4}$$

Providing for the possibility of episode splitting by using the conditional survivor function, the log-likelihood is

$$\ell = \sum_{i\in\mathcal{E}} \log\left\{\frac{V(z_{t_i})}{b\,(t_i - c)}\right\} + \sum_{i\in\mathcal{N}} \log\left\{\frac{Q(z_{t_i})}{Q(z_{s_i})}\right\} \tag{5}$$

I. The first and second derivatives for the A and B-terms of the model may then be written as

$$\frac{\partial \ell}{\partial \alpha_{j_a}} = \sum_{i \in \mathcal{E}} \frac{1}{b} \left\{ z_{t_i} - V(z_{t_i}) \right\} A_{i,j_a} + \sum_{i \in \mathcal{N}} \frac{1}{b} \left\{ V(z_{t_i}) - V(z_{s_i}) \right\} A_{i,j_a}$$

$$\frac{\partial \ell}{\partial \beta_{j_b}} = \sum_{i \in \mathcal{E}} \left\{ z_{t_i} \left[z_{t_i} - V(z_{t_i}) \right] - 1 \right\} B_{i,j_b} + \sum_{i \in \mathcal{N}} \left\{ z_{t_i} V(z_{t_i}) - z_{s_i} V(z_{s_i}) \right\} B_{i,j_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \alpha_{k_a}} = \sum_{i \in \mathcal{E}} \frac{1}{b^2} \left\{ V(z_{t_i}) \left[V(z_{t_i}) - z_{t_i} \right] - 1 \right\} A_{i,j_a} A_{i,k_a} + \sum_{i \in \mathcal{N}} \frac{1}{b^2} \left\{ V(z_{t_i}) \left[z_{t_i} - V(z_{t_i}) \right] - V(z_{s_i}) \left[z_{s_i} - V(z_{s_i}) \right] \right\} A_{i,j_a} A_{i,k_a}$$

$$\frac{\partial^2 \ell}{\partial \beta_{j_b} \partial \beta_{k_b}} = \sum_{i \in \mathcal{E}} \left\{ z_{t_i} V(z_{t_i}) \left[1 + z_{t_i} (V(z_{t_i}) - z_{t_i}) \right] - 2 z_{t_i}^2 \right\} B_{i,j_b} B_{i,k_b} + \sum_{i \in \mathcal{N}} \left\{ z_{t_i} V(z_{t_i}) \left[z_{t_i} (z_{t_i} - V(z_{t_i})) - 1 \right] - z_{s_i} V(z_{s_i}) \left[z_{s_i} (z_{s_i} - V(z_{s_i})) - 1 \right] \right\} B_{i,j_b} B_{i,k_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \beta_{k_b}} = \sum_{i \in \mathcal{E}} \frac{1}{b} \left\{ (1 + z_{t_i} V(z_{t_i})) \left[V(z_{t_i}) - z_{t_i} \right] - z_{t_i} \right\} A_{i,j_a} B_{i,k_b} + \sum_{i \in \mathcal{N}} \frac{1}{b} \left\{ V(z_{t_i}) \left[z_{t_i} \left(z_{t_i} - V(z_{t_i}) \right) - 1 \right] - V(z_{s_i}) \left[z_{s_i} \left(z_{s_i} - V(z_{s_i}) \right) - 1 \right] \right\} A_{i,j_a} B_{i,k_b}$$

II. In the case of an extended model, if $c > 0$, one also needs the derivatives with respect to the γ coefficients. They can be written as follows.

$$\frac{\partial \ell}{\partial \gamma_{j_a}} = \sum_{i \in \mathcal{E}} \frac{c}{b \left(t_i - c \right)} \left\{ b + z_{t_i} - V(z_{t_i} \right\} C_{i,j_a} + \sum_{i \in \mathcal{N}} \frac{c}{b \left(t_i - c \right)} V(z_{t_i}) \, C_{i,j_a}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \gamma_{k_a}} = \sum_{i \in \mathcal{E}} \frac{c}{b^2 (t_i - c)} \left\{ V(z_{t_i}) \left[V(z_{t_i}) - z_{t_i} \right] - 1 \right\} A_{i,j_a} C_{i,k_a} +$$
$$\sum_{i \in \mathcal{N}} \frac{c}{b^2} \left\{ \frac{V(z_{t_i}) \left[z_{t_i} - V(z_{t_i}) \right]}{t_i - c} - \frac{V(z_{s_i}) \left[z_{s_i} - V(z_{s_i}) \right]}{s_i - c} \right\} A_{i,j_a} C_{i,}$$

$$\frac{\partial^2 \ell}{\partial \beta_{j_b} \partial \gamma_{k_b}} = \sum_{i \in \mathcal{E}} \frac{c}{b (t_i - c)} \left\{ V(z_{t_i}) \left[1 + z_{t_i} (V(z_{t_i}) - z_{t_i}) \right] - 2 z_{t_i} \right\} B_{i,j_b} C_{i,k_b}$$
$$\sum_{i \in \mathcal{N}} \frac{c}{b} \left\{ \frac{V(z_{t_i})}{t_i - c} \left[z_{t_i} (z_{t_i} - V(z_{t_i})) - 1 \right] - \right.$$
$$\left. \frac{V(z_{s_i})}{s_i - c} \left[z_{s_i} (z_{s_i} - V(z_{s_i})) - 1 \right] \right\} B_{i,j_b} C_{i,k_b}$$

$$\frac{\partial^2 \ell}{\partial \gamma_{j_a} \partial \gamma_{k_b}} = \sum_{i \in \mathcal{E}} \frac{c}{b (t_i - c)^2} \left\{ t_i \left[b + z_{t_i} - V(z_{t_i}) \right] - \right.$$
$$\left. \frac{c}{b} \left(1 - V(z_{t_i}) \left[V(z_{t_i}) - z_{t_i} \right] \right) \right\} C_{i,j_a} C_{i,k_b} +$$
$$\sum_{i \in \mathcal{N}} \frac{c}{b} \left\{ \frac{V(z_{t_i})}{(t_i - c)^2} \left[t_i - \frac{c}{b} \left(V(z_{t_i}) - z_{t_i} \right) \right] - \right.$$
$$\left. \frac{V(z_{s_i})}{(s_i - c)^2} \left[s_i - \frac{c}{b} \left(V(z_{s_i}) - z_{s_i} \right) \right] \right\} C_{i,j_a} C_{i,k_b}$$

Starting Values. In most situations, estimating standard log-normal models does not need very accurate initial estimates. Therefore as default starting values used by TDA all model coefficients are set to zero, with the exception of the constant in the A-term of the model with is set to the negative of the logarithm of the constant rate of an accordingly defined exponential null model. On the other hand, initial estimates for the extended log-normal model *are* critical, in particular the estimates for the C-term of the model. This is implied by the shift operation done with the c parameter. However, simple estimates of useful starting values are not available, and therefore TDA uses as a default the same starting values as with the standard log-normal model, and in addition sets all C-term coefficients to zero. In many cases this will result in convergence difficulties, in particular when episode splitting is applied. One way out of such difficulties is first to estimate a standard log-normal model, and then to use the resulting parameter estimates as starting values for the

Box 1 Output from command file `rt12.cf`

```
Model: Log-normal

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Number of model parameters: 5
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Log-likelihood of exponential null model: -2514.02
Scaling factor for log-likelihood: -0.001
Using default starting values.

Convergence reached in 6 iterations.
Number of function evaluations: 7 (7,7)

Maximum of log likelihood: -2437.52
Norm of final gradient vector: 0.000491553
Last absolute change of function value: 8.24072e-08
Last relative change in parameters: 0.00239838

Idx SN Org Des MT Variable      Coeff      Error    C/Error  Signif
-----------------------------------------------------------------------
  1  1   0   1  A Constant     4.3688     0.0960    45.5055  1.0000
  2  1   0   1  A COHO2       -0.4608     0.1255    -3.6707  0.9998
  3  1   0   1  A COHO3       -0.6110     0.1272    -4.8036  1.0000
  4  1   0   1  A W           -0.3821     0.1062    -3.5992  0.9997
  5  1   0   1  B Constant     0.2036     0.0342     5.9508  1.0000

Log likelihood (starting values): -2603.1677
Log likelihood (final estimates): -2437.5243
```

extended model, with the additional C-term coefficients set to zero or, if available, to some better estimates.

Example 1 The standard log-normal model is similar to the log-logistic model. One can see this by comparing the results from command file `rt12.cf`, shown in Box 1 with the estimation results in **6.17.3.5**.

Generalized Residuals. For the log-normal model, one can use the `pres` option to request a table with generalized residuals.

Printing Estimated Rates. The `prate` option can be used to request tables with estimated rates, survivor and density functions, depending on selected covariate constellations. The `rrisk` option cannot be used with the log-normal model.

6.17.3.7 Generalized Gamma Models

This section describes a model where, in the single transition case, the duration is assumed to follow a generalized gamma distribution. The model is useful since it contains as special cases the exponential, the Weibull, and the log-normal models. Therefore it can be used to discriminate between these models.

We follow the explanation of this model given by Lawless ([1982], p. 237). Lawless shows that, if the duration variable T has a generalized gamma distribution, then by re-parameterization, $\log(T)$ can be expressed as

$$\log(T) = a + b\,W \qquad b > 0$$

where W has density function

$$f_w(w) = \frac{\kappa^{\kappa-1/2}}{\Gamma(\kappa)} \exp\left\{\sqrt{\kappa}\,w - \kappa\,\exp\left(\frac{w}{\sqrt{\kappa}}\right)\right\} \qquad \kappa > 0$$

$\Gamma(\kappa)$ is the gamma function. Now, given this re-parameterization, the density of T can be written as

$$f(t\,;a,b,\kappa) = \frac{\kappa^{\kappa-1/2}}{b\,t\,\Gamma(\kappa)} \exp\left\{\sqrt{\kappa}\,z_t - q_t\right\}$$

$$z_t = \frac{\log(t) - a}{b}$$

$$q_t = \kappa\,\exp\left(\frac{z_t}{\sqrt{\kappa}}\right)$$

Of course, the density is only defined for $t > 0$. The associated cumulative distribution function, $F(t)$, can be derived as the integral of the density to be

$$F(t\,;a,b,\kappa) = I(q_t,\kappa) = \int_0^{q_t} \frac{\tau^{\kappa-1}}{\Gamma(\kappa)} \exp(-\tau)\,d\tau$$

$I(q_t,\kappa)$ is called the incomplete gamma integral. Using this expression, the survivor function is

$$G(t\,;a,b,\kappa) = 1 - I(q_t,\kappa)$$

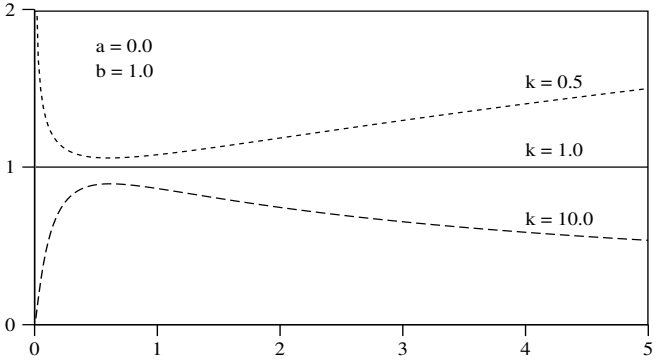


Figure 1 Gamma transition rates. Plot created with command file `plot-g1.cf`.

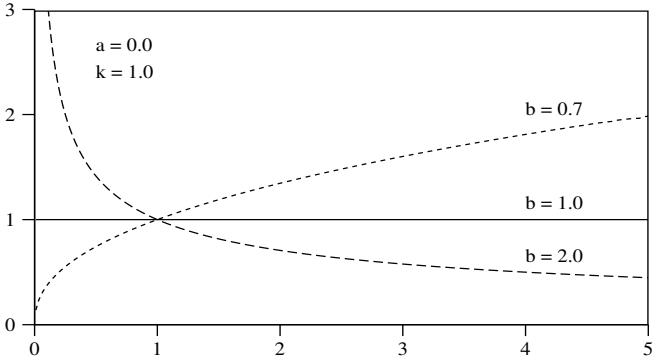


Figure 2 Gamma transition rates. Plot created with command file `plot-g2.cf`.

and the transition rate for this model can finally be written as

$$r(t\,;a,b,\kappa) = \frac{f(t\,;a,b,\kappa)}{G(t\,;a,b,\kappa)} = \frac{\kappa^{\kappa-1/2}\exp\{\sqrt{\kappa}\,z_t - q_t\}}{b\,t\,\Gamma(\kappa)\,(1-I(q_t,\kappa))}$$

The model has three parameters, a, b, and κ. a can take arbitrary values, b and κ must be positive. Special cases are the exponential model, if $b = 1$ and $\kappa = 1$; the Weibull model, if $\kappa = 1$; and a log-normal model is reached if $\kappa \to \infty$. Figures 1 and 2 show graphs of the rate function for some parameter constellations.

Implementation. The generalized gamma model has three parameters to be estimated. However, the current implementation in TDA follows a proposal of Lawless to estimate only two of them, a and b, by maximum likelihood, and to treat the κ-parameter as externally given. This very much simplifies the calculations. The model approach implemented in TDA may then be written as

$$
\begin{aligned}
r_{jk}(t) &= \frac{\kappa^{\kappa-1/2}\exp\left\{\sqrt{\kappa}\, z_t^{(jk)} - q_t^{(jk)}\right\}}{b_{jk}\, t\, \Gamma(\kappa)\,(1 - I(q_t^{(jk)}, \kappa))} \qquad (1)\\
z_t^{(jk)} &= \frac{\log(t) - a_{jk}}{b_{jk}}\\
q_t^{(jk)} &= \kappa \exp\left(\frac{z_t^{(jk)}}{\sqrt{\kappa}}\right)\\
a_{jk} &= A^{(jk)}\alpha^{(jk)}\\
b_{jk} &= \exp\left(B^{(jk)}\beta^{(jk)}\right)\\
\kappa &> 0 \quad \text{externally given}
\end{aligned}
$$

It is assumed that the first component of each of the covariate (row) vectors $A^{(jk)}$ and $B^{(jk)}$ is a constant equal to one. The associated coefficient vectors, $\alpha^{(jk)}$ and $\beta^{(jk)}$, are the model parameters to be estimated. κ is assumed to be given externally, so there are in effect only two model terms.

The generalized gamma model has model number 13. The κ coefficient can be specified with the parameter

`kgam` = κ,

Default is `kgam=1`. Note that this coefficient must be positive.

Maximum Likelihood Estimation. The model is estimated following the outline given in **6.17.1.2** and **6.17.1.3**. To simplify the notation we omit indices for transitions and use the abbreviations defined in (1). Providing for the possibility of episode splitting by using the conditional

survivor function the log-likelihood is

$$\ell = \sum_{i \in \mathcal{E}} \log \left\{ \frac{\kappa^{\kappa - 1/2}}{b\, t\, \Gamma(\kappa)} \exp\left(\sqrt{\kappa}\, z_t - q_t\right) \right\} + \sum_{i \in \mathcal{Z}} \log\left\{1 - I(q_t, \kappa)\right\} - \sum_{i \in \mathcal{N}} \log\left\{1 - I(q_s, \kappa)\right\}$$

Using the notation

$$Q_t = \frac{q_t^{\kappa} \exp(-q_t)}{\sqrt{\kappa}\, \Gamma(\kappa)\, [1 - I(q_t, \kappa)]}$$
$$Q_s = \frac{q_s^{\kappa} \exp(-q_s)}{\sqrt{\kappa}\, \Gamma(\kappa)\, [1 - I(q_s, \kappa)]}$$

the first and second derivatives with respect to the α- and β-coefficients are

$$\frac{\partial \ell}{\partial \alpha_{j_a}} = \sum_{i \in \mathcal{E}} \left\{ \frac{q_t}{\sqrt{\kappa}} - \sqrt{\kappa} \right\} \frac{1}{b}\, A_{i,j_a} + \sum_{i \in \mathcal{Z}} \frac{Q_t}{b}\, A_{i,j_a} - \sum_{i \in \mathcal{N}} \frac{Q_s}{b}\, A_{i,j_a}$$

$$\frac{\partial \ell}{\partial \beta_{j_b}} = \sum_{i \in \mathcal{E}} \left\{ \left(\frac{q_t}{\sqrt{\kappa}} - \sqrt{\kappa} \right) z_t - 1 \right\} B_{i,j_b} + \sum_{i \in \mathcal{Z}} Q_t\, z_t\, B_{i,j_b} - \sum_{i \in \mathcal{N}} Q_s\, z_s\, B_{i,j_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \alpha_{k_a}} = \sum_{i \in \mathcal{E}} -\frac{q_t}{\kappa\, b^2}\, A_{i,j_a} A_{i,k_a} + \sum_{i \in \mathcal{Z}} \left\{ \frac{q_t}{\sqrt{\kappa}} - \sqrt{\kappa} - Q_t \right\} \frac{Q_t}{b^2}\, A_{i,j_a} A_{i,k_a} - \sum_{i \in \mathcal{N}} \left\{ \frac{q_s}{\sqrt{\kappa}} - \sqrt{\kappa} - Q_s \right\} \frac{Q_s}{b^2}\, A_{i,j_a} A_{i,k_a}$$

$$\begin{aligned}
\frac{\partial^2 \ell}{\partial \beta_{j_b} \partial \beta_{k_b}} = & \sum_{i \in \mathcal{E}} \left\{ \sqrt{\kappa} - \left(\frac{z_t}{\sqrt{\kappa}} + 1 \right) \frac{q_t}{\sqrt{\kappa}} \right\} z_t \, B_{i,j_b} B_{i,k_b} + \\
& \sum_{i \in \mathcal{Z}} \left\{ \left(\frac{q_t}{\sqrt{\kappa}} - \sqrt{\kappa} - Q_t \right) z_t - 1 \right\} Q_t \, z_t \, B_{i,j_b} B_{i,k_b} - \\
& \sum_{i \in \mathcal{N}} \left\{ \left(\frac{q_s}{\sqrt{\kappa}} - \sqrt{\kappa} - Q_s \right) z_s - 1 \right\} Q_s \, z_s \, B_{i,j_b} B_{i,k_b} \\
\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \beta_{k_b}} = & \sum_{i \in \mathcal{N}} \left\{ \sqrt{\kappa} - \left(\frac{z_t}{\sqrt{\kappa}} + 1 \right) \frac{q_t}{\sqrt{\kappa}} \right\} \frac{1}{b} \, A_{i,j_a} B_{i,k_b} + \\
& \sum_{i \in \mathcal{Z}} \left\{ \left(\frac{q_t}{\sqrt{\kappa}} - \sqrt{\kappa} - Q_t \right) z_t - 1 \right\} \frac{Q_t}{b} \, A_{i,j_a} B_{i,k_b} - \\
& \sum_{i \in \mathcal{N}} \left\{ \left(\frac{q_s}{\sqrt{\kappa}} - \sqrt{\kappa} - Q_s \right) z_s - 1 \right\} \frac{Q_s}{b} \, A_{i,j_a} B_{i,k_b}
\end{aligned}$$

Example 1 To illustrate, we use command file `rt13.cf`. The parameter estimates are shown in Box 1.

Generalized Residuals. For the gamma models, one can use the `pres` option to request a table with generalized residuals.

Printing Estimated Rates. The `prate` option can be used to request tables with estimated rates, survivor and density functions, depending on selected covariate constellations. Note that, for $t = 0$, the rate and density is always set to zero, and the survivor function is set to 1; this might not be correct. The `rrisk` option cannot be used with the gamma model.

Box 1 Output from command file `rt13.cf`

```
Model: Generalized Gamma (kgam=1).

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Number of model parameters: 5
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Log-likelihood of exponential null model: -2514.02
Scaling factor for log-likelihood: -0.001
Using default starting values.

Convergence reached in 6 iterations.
Number of function evaluations: 7 (7,7)

Maximum of log likelihood: -2473.54
Norm of final gradient vector: 9.1704e-08
Last absolute change of function value: 2.03216e-12
Last relative change in parameters: 4.57906e-05

Idx SN Org Des MT Variable      Coeff      Error    C/Error  Signif
-------------------------------------------------------------------
  1  1   0   1  A Constant     5.0055     0.0906    55.2452  1.0000
  2  1   0   1  A COHO2       -0.5339     0.1201    -4.4448  1.0000
  3  1   0   1  A COHO3       -0.6671     0.1235    -5.4003  1.0000
  4  1   0   1  A W           -0.5191     0.1012    -5.1274  1.0000
  5  1   0   1  B Constant     0.0700     0.0366     1.9119  0.9441

Log likelihood (starting values): -2514.0199
Log likelihood (final estimates): -2473.5432
```

6.17.3.8 Inverse Gaussian Models

This section describes the Inverse Gaussian model introduced for duration analysis by Chhikara and Folks [1977]. We follow the description given by Lancaster ([1990], p. 49). In the single transition case the model is derived by assuming that the duration variable, T, follows an Inverse Gaussian distribution with density function

$$f(t) = \frac{1}{b\,t^{3/2}}\,\phi\left(\frac{a\,t-1}{b\,\sqrt{t}}\right) \qquad b>0 \tag{1}$$

As in 6.17.3.6, ϕ and Φ are used, respectively, to denote the standard normal density and distribution function. The model has two parameters. The parameter a can take arbitrary values, the parameter b generates dispersion and should be positive. The survivor function corresponding to (1) is

$$G(t) = \Phi\left(\frac{1-a\,t}{b\,\sqrt{t}}\right) - \exp\left(\frac{2\,a}{b^2}\right)\,\Phi\left(\frac{-1-a\,t}{b\,\sqrt{t}}\right)$$

Figure 1 shows graphs of the rate function, defined as $r(t) = f(t)/G(t)$, for $b = 1$ and some values of a. As is seen, the graphs are similar to log-normal rates.

Implementation. The Inverse Gaussian model has two parameters, so there are two possibilities to include covariates. Similar to the log-normal model, TDA uses a linear link function for the parameter a, and additionally provides the possibility to link covariates to the dispersion parameter b via an exponential link function. So one gets the following model formulation for the transition rate from origin state j to destination state k.

$$r_{jk}(t) = \frac{\frac{1}{b_{jk}\,t^{3/2}}\,\phi\left(\frac{a_{jk}\,t-1}{b_{jk}\,\sqrt{t}}\right)}{\Phi\left(\frac{1-a_{jk}\,t}{b_{jk}\,\sqrt{t}}\right) - \exp\left(\frac{2\,a_{jk}}{b_{jk}^2}\right)\,\Phi\left(\frac{-1-a_{jk}\,t}{b_{jk}\,\sqrt{t}}\right)}$$

$$a_{jk} = A^{(jk)}\alpha^{(jk)}$$

$$b_{jk} = \exp\left\{B^{(jk)}\beta^{(jk)}\right\}$$

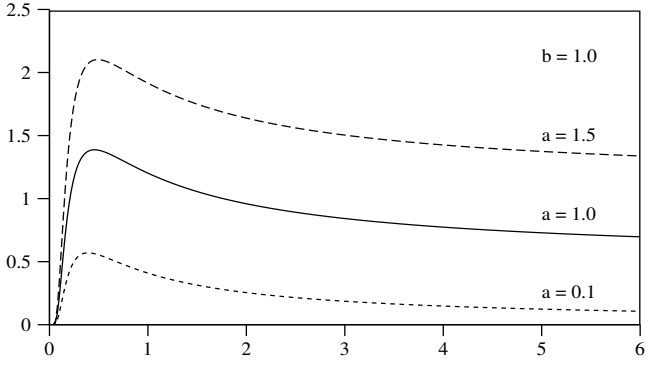


Figure 1 Inverse Gaussian transition rates. Plot created with command file `plot-ig.cf`.

It is assumed that the first component of each of the covariate (row) vectors $A^{(jk)}$ and $B^{(jk)}$ is a constant equal to one. The associated coefficient vectors $\alpha^{(jk)}$ and $\beta^{(jk)}$ are the model parameters to be estimated. The transition rate model with inverse Gaussian duration distribution has model number 14.

Maximum Likelihood Estimation. The model is estimated following the outline given in **6.17.1.2** and **6.17.1.3**. To simplify the formulas we omit indices for transitions and use the following abbreviations:

$$z_1(t) = \frac{a\,t - 1}{b\,\sqrt{t}} \qquad z_2(t) = \frac{a\,t + 1}{b\,\sqrt{t}} \qquad E = \exp\left\{\frac{2\,a}{b^2}\right\}$$

$$f(t) = \frac{1}{b\,t^{3/2}}\,\phi\left(z_1(t)\right)$$

$$f_a(t) = -\frac{z_1(t)\,\sqrt{t}}{b}\,f(t)$$

$$f_b(t) = \left[z_1(t)^2 - 1\right]\,f(t)$$

$$f_{aa}(t) = \left[z_1(t)^2 - 1\right]\,\frac{t}{b^2}\,f(t)$$

$$f_{bb}(t) = \left[z_1(t)^2 - 1\right]\,f_b(t) - 2\,z_1(t)^2\,f(t)$$

$$f_{ab}(t) = \frac{z_1(t)\,\sqrt{t}}{b}\,\{2\,f(t) - f_b(t)\}$$

$$G(t) = \Phi\left(-z_1(t)\right) - E\,\Phi\left(-z_2(t)\right)$$

$$G_a(t) = \frac{\sqrt{t}}{b}\left\{E\,\phi\left(z_2(t)\right) - \phi\left(z_1(t)\right)\right\} - \frac{2\,E}{b^2}\,\Phi\left(-z_2(t)\right)$$

$$G_b(t) = z_1(t)\,\phi\left(z_1(t)\right) + E\left\{\frac{4\,a}{b^2}\,\Phi\left(-z_2(t)\right) - z_2(t)\,\phi\left(z_2(t)\right)\right\}$$

$$G_{aa}(t) = \frac{4}{b^2}\,E\left\{\frac{\sqrt{t}}{b}\,\phi\left(z_2(t)\right) - \frac{1}{b^2}\,\Phi\left(-z_2(t)\right)\right\} + \frac{t}{b^2}\left\{z_1(t)\,\phi\left(z_1(t)\right) - z_2(t)\,E\,\phi\left(z_2(t)\right)\right\}$$

$$G_{bb}(t) = z_1(t)\left[z_1(t)^2 - 1\right]\,\phi\left(z_1(t)\right) + z_2(t)\left[1 - z_2(t)^2 + \frac{8\,a}{b^2}\right]E\,\phi\left(z_2(t)\right) - \left[1 + \frac{2\,a}{b^2}\right]\frac{8\,a}{b^2}\,E\,\Phi\left(-z_2(t)\right)$$

$$G_{ab}(t) = \frac{\sqrt{t}}{b}\left\{\left[1 - z_1(t)^2\right]\,\phi\left(z_1(t)\right)\left[1 - z_2(t)^2 + \frac{4\,a}{b^2}\right]E\,\phi\left(z_2(t)\right)\right\} + \frac{2}{b^2}\,E\left\{\left[2 + \frac{4\,a}{b^2}\right]\,\Phi\left(-z_2(t)\right) - z_2(t)\,\phi\left(z_2(t)\right)\right\}$$

Using these abbreviations and providing for the possibility of episode splitting by using conditional survivor functions, the log-likelihood is

$$\ell = \sum_{i\in\mathcal{E}} \log\left(f(t_i)\right) + \sum_{i\in\mathcal{Z}} \log\left(G(t_i)\right) - \sum_{i\in\mathcal{N}} \log\left(G(s_i)\right)$$

$\mathcal{E}$, $\mathcal{Z}$, and $\mathcal{N}$, are the sets of episodes with an event, of censored episodes, and of all episodes, respectively. The subscript i indexes single episodes with starting time s_i and ending time t_i. Omitting terms for starting times, the first and second derivatives are

$$\frac{\partial \ell}{\partial \alpha_{j_a}} = \sum_{i\in\mathcal{E}} \frac{f_a(t_i)}{f(t_i)} A_{i,j_a} + \sum_{i\in\mathcal{Z}} \frac{G_a(t_i)}{G(t_i)} A_{i,j_a}$$

$$\frac{\partial \ell}{\partial \beta_{j_b}} = \sum_{i \in \mathcal{E}} \frac{f_b(t_i)}{f(t_i)} B_{i,j_b} + \sum_{i \in \mathcal{Z}} \frac{G_b(t_i)}{G(t_i)} B_{i,j_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \alpha_{k_a}} = \sum_{i \in \mathcal{E}} \left\{ \frac{f_{aa}(t_i)}{f(t_i)} - \left[\frac{f_a(t_i)}{f(t_i)} \right]^2 \right\} A_{i,j_a} A_{i,k_a} + \sum_{i \in \mathcal{Z}} \left\{ \frac{G_{aa}(t_i)}{G(t_i)} - \left[\frac{G_a(t_i)}{G(t_i)} \right]^2 \right\} A_{i,j_a} A_{i,k_a}$$

$$\frac{\partial^2 \ell}{\partial \beta_{j_b} \partial \beta_{k_b}} = \sum_{i \in \mathcal{E}} \left\{ \frac{f_{bb}(t_i)}{f(t_i)} - \left[\frac{f_b(t_i)}{f(t_i)} \right]^2 \right\} B_{i,j_b} B_{i,k_b} + \sum_{i \in \mathcal{Z}} \left\{ \frac{G_{bb}(t_i)}{G(t_i)} - \left[\frac{G_b(t_i)}{G(t_i)} \right]^2 \right\} B_{i,j_b} B_{i,k_b}$$

$$\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \beta_{k_b}} = \sum_{i \in \mathcal{E}} \left\{ \frac{f_{ab}(t_i)}{f(t_i)} - \frac{f_a(t_i)}{f(t_i)} \frac{f_b(t_i)}{f(t_i)} \right\} A_{i,j_a} B_{i,k_b} + \sum_{i \in \mathcal{Z}} \left\{ \frac{G_{ab}(t_i)}{G(t_i)} - \frac{G_a(t_i)}{G(t_i)} \frac{G_b(t_i)}{G(t_i)} \right\} A_{i,j_a} B_{i,k_b}$$

Starting Values. For models which are not too complicated, maximum likelihood estimation seems to be relatively robust with respect to initial estimates. TDA takes the following default values: The constant of the A-term of the model is taken to be the rate of an accordingly defined exponential model, the constant of the B-term of the model is defined as the logarithm of the square root of this rate. All other coefficients are set to zero.

Example 1 As an illustration we use the main example data set. The command file is `rt14.cf`, estimation results are shown in Box 1.

Generalized Residuals. For the inverse Gaussian model, one can use the `pres` option to request a table with generalized residuals.

Printing Estimated Rates. The `prate` option can be used to request tables with estimated rates, survivor and density functions, depending on selected covariate constellations. The `rrisk` option cannot be used with the inverse Gaussian model.

Box 1 Output from command file `rt14.cf`

```
Model: Inverse Gaussian

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Number of model parameters: 5
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Log-likelihood of exponential null model: -2514.02
Scaling factor for log-likelihood: -0.001
Using default starting values.

Convergence reached in 6 iterations.
Number of function evaluations: 7 (7,7)

Maximum of log likelihood: -2434.34
Norm of final gradient vector: 9.30888e-05
Last absolute change of function value: 8.96273e-10
Last relative change in parameters: 0.00415467

Idx SN Org Des MT Variable      Coeff      Error    C/Error  Signif
-------------------------------------------------------------------
  1  1   0   1  A Constant     0.0007     0.0015     0.4225  0.3273
  2  1   0   1  A COHO2        0.0067     0.0024     2.8243  0.9953
  3  1   0   1  A COHO3        0.0088     0.0026     3.4284  0.9994
  4  1   0   1  A W            0.0093     0.0021     4.4894  1.0000
  5  1   0   1  B Constant    -1.7092     0.0311   -55.0104  1.0000

Log likelihood (starting values): -2699.6178
Log likelihood (final estimates): -2434.3446
```

6.17.4 Mixture Models

This section deals with transition rate models where the distribution of the waiting times for an event is a mixture distribution. Currently we have only a single subsection that describes parametric models with a gamma mixing distribution.

6.17.4.1 Gamma Mixture Models

6.17.4.1 Gamma Mixture Models

Introduction

It is assumed that the transition rate, $r(t|x,v)$, depends on a vector, x, of observed time-invariant variables and on a scalar stochastic term, v, which is not observed. The question is how the variables comprised in x influence the transition rate, but this can not be observed directly. In any observed sample of episodes, the transition rate depends also on the values of the stochastic term v.

The same is true for the associated density and survivor functions. The relation between transition rate, density and survivor function must therefore be written as

$$r(t \mid x,v) = \frac{f(t \mid x,v)}{G(t \mid x,v)} \tag{1}$$

$$G(t \mid x,v) = \exp\left(-\int_0^t r(\tau \mid x,v)\,d\tau\right) \tag{2}$$

To find an estimation approach, we shall make some simplifying assumptions. First, that the transition rate can be expressed as

$$r(t \mid x,v) = r^u(t \mid x)\,v \qquad v \geq 0 \tag{3}$$

In fact, we assume that the component $r^u(t \mid x)$ of this expression, called the *underlying* rate, is parametrically given according to one of the models discussed in **6.17.3**. The cumulative transition rate may then be written as

$$H(t \mid x,v) = v\,H^u(t \mid x) = v\int_0^t r^u(\tau \mid x)\,d\tau \tag{4}$$

The second basic assumption is that the stochastic term, v, follows a gamma distribution with expectation $\mathrm{E}(v) = 1$. Implied by this assumption, the density function of v can be written as

$$f_v(v) = \frac{\kappa^\kappa\, v^{\kappa-1}}{\Gamma(\kappa)}\exp(-\kappa\,v) \qquad \kappa > 0$$

The variance of this distribution is $\mathrm{Var}(v) = 1/\kappa$. The next step is to calculate the resulting mixture distribution for the observed durations. First, for the density and survivor functions, one gets

$$\begin{aligned} f(t \mid x) &= \int_0^\infty f(t \mid x, v)\, f_v(v)\, dv \\ G(t \mid x) &= \int_0^\infty G(t \mid x, v)\, f_v(v)\, dv \end{aligned}$$

The mixtures are expectations according to the distribution of v. The calculation is easy because the gamma distribution implies the basic equality

$$\int_0^\infty \exp(-v\, s(t))\, f_v(v)\, dv = \left[1 + \frac{1}{\kappa}\, s(t)\right]^{-\kappa} \tag{5}$$

which holds for any real valued function $s(t)$ (see, e.g., Lancaster [1990, p. 328]). Therefore, using (2) and (4), the unconditional survivor function is

$$G(t \mid x) = \int_0^\infty \exp\left(-v\, H^u(t \mid x)\right)\, f_v(v)\, dv = \left[1 + \frac{1}{\kappa} H^u(t \mid x)\right]^{-\kappa} \tag{6}$$

The unconditional density function can be found by differentiating the negative value of the unconditional survivor function, resulting in

$$f(t \mid x) = r^u(t \mid x) \left[1 + \frac{1}{\kappa} H^u(t \mid x)\right]^{-\kappa-1} \tag{7}$$

Finally, the unconditional transition rate is

$$r(t \mid x) = \frac{f(t \mid x)}{G(t \mid x)} = r^u(t \mid x) \left[1 + \frac{1}{\kappa} H^u(t \mid x)\right]^{-1} \tag{8}$$

Note that the unconditional transition rate cannot be derived directly as an expectation according to the mixing distribution since, in general,

$$r(t \mid x) \neq \int_0^\infty r(t \mid x, v)\, f_v(v)\, dv$$

Also, there is no linear relation between $r(t \mid x)$ and $r^u(t \mid x)$. Instead (cf. Lancaster [1990, p. 63f]), the relation (8) may be expressed as

$$r(t \mid x) = r^u(t \mid x)\, \mathrm{E}(v \mid x, T \geq t)$$

where $\mathrm{E}(v \mid x, T \geq t)$ is the expectation of v for the part of the population which stays in the origin state at least until t. This expectation is decreasing with t if the variance of the mixing distribution is positive. For example, the underlying rate $r^u(t \mid x)$ may be assumed to be time-independent, a standard exponential model, but if there is unobserved heterogeneity, this is compatible with an decreasing observed rate $r(t \mid x)$.

Several Transitions. Extension to a situation with several transitions is straightforward. To simplify notation, we omit dependence on covariates. Following the basic idea formulated in (3), for each transition from origin state $j \in \mathcal{O}$ to destination state $k \in \mathcal{D}_j$, transition-specific rates can be defined by

$$r_{jk}(t \mid v_{jk}) = r^u_{jk}(t)\, v_{jk} \qquad v_{jk} \geq 0 \tag{9}$$

$r^u_{jk}(t)$ is the underlying rate used for model formulation. The stochastic term v_{jk}, to provide for the possibility of unobserved heterogeneity, is assumed to be specific for each transition, and stochastically independent across transitions. And as before, it is assumed that these stochastic terms are gamma distributed with unit expectation. The overall exit rate from origin state j is then

$$r_j(t \mid v_j) = \sum_{k \in \mathcal{D}_j} r^u_{jk}(t)\, v_{jk} \tag{10}$$

where v_j is a vector of the stochastic terms v_{jk}, $k \in \mathcal{D}_j$. To simplify notation we also define

$$H^u_{jk}(t) = \int_0^t r^u_{jk}(\tau)\, d\tau$$

The next step is to define transition-specific sub-density functions $\tilde{f}_{jk}(t \mid v_{jk})$, and the overall survivor function $G_j(t \mid v_j)$, in order to express transition-specific rates as

$$r_{jk}(t \mid v_{jk}) = \frac{\tilde{f}_{jk}(t \mid v_{jk})}{G_j(t \mid v_j)} \tag{11}$$

Then, analogously to the single transition case, one has to derive the mixture distributions describing the data on the observational level. First, to derive the unconditional survivor function $G_j(t)$, one can take the expectation of $G_j(t \mid v_j)$ over all components of the vector v_j. This is easy

because we have assumed transition-specific stochastic terms. Therefore, using (10), one finds

$$G_j(t \mid v_j) = \prod_{k \in \mathcal{D}_j} \exp\left(-v_{jk} \int_0^t r^u_{jk}(\tau)\, d\tau\right) \tag{12}$$

Obviously, the expectation can be taken for each term separately, and using (5), one gets

$$G_j(t) = \prod_{k \in \mathcal{D}_j} \left[1 + \frac{1}{\kappa_{jk}} H^u_{jk}(t)\right]^{-\kappa_{jk}} \tag{13}$$

which may be written as

$$G_j(t) = \prod_{k \in \mathcal{D}_j} \tilde{G}_{jk}(t) \quad \text{where} \quad \tilde{G}_{jk}(t) = \left[1 + \frac{1}{\kappa_{jk}} H^u_{jk}(t)\right]^{-\kappa_{jk}} \tag{14}$$

Secondly, one has to derive the expectation of $\tilde{f}_{jk}(t \mid v_{jk})$. This can be done by starting with (9) and (11), resulting in

$$\tilde{f}_{jk}(t) = r^u_{jk}(t)\, v_{jk}\, G_j(t \mid v_j)$$

Using then (12), the expectation of the right-hand side is a product of transition-specific expectations. Except for the transition (j,k), the expectation is directly given by (5). The kth term contains v_{jk}, but differentiating the mixing integral with respect to H^u_{jk} gives

$$\int_0^\infty v_{jk} \exp(-v_{jk}\, H^u_{jk}(t))\, f_{v_{jk}}(v_{jk})\, dv_{jk} = \left[1 + \frac{1}{\kappa_{jk}} H^u_{jk}(t)\right]^{-\kappa_{jk}-1}$$

The expectation of $\tilde{f}_{jk}(t \mid v_{jk})$ is therefore given by

$$\tilde{f}_{jk}(t) = r^u_{jk}(t) \left[1 + \frac{1}{\kappa_{jk}} H^u_{jk}(t)\right]^{-1} \prod_{k \in \mathcal{D}_j} \left[1 + \frac{1}{\kappa_{jk}} H^u_{jk}(t)\right]^{-\kappa_{jk}} \tag{15}$$

Finally, taking together (13) and (15), the unconditional transition-specific rates are

$$r_{jk}(t) = \frac{\tilde{f}_{jk}(t)}{G_j(t)} = r^u_{jk}(t) \left[1 + \frac{1}{\kappa_{jk}} H^u_{jk}(t)\right]^{-1} \tag{16}$$

Maximum Likelihood Estimation. The approach to derive transition rate models with an additional gamma distributed stochastic term, discussed in the previous section, can be summarized as follows. One starts with a parametrically given transition rate, as in (3), and assumes that the unobserved heterogeneity is gamma distributed with unit mean and variance $1/\kappa$. Then one derives the observed mixture distribution, described by $f(t \mid x)$, $G(t \mid x)$, and $r(t \mid x)$, given by (6), (7), and (8), respectively.

It follows that the log-likelihood can be set up in the usual way. First, for a single transition (with $\mathcal{E}$ and $\mathcal{N}$ denoting the sets of episodes with an event, and of censored episodes, respectively) the log-likelihood may be written as

$$\begin{aligned} \ell &= \sum_{i\in\mathcal{E}} \log(r(t_i)) + \sum_{i\in\mathcal{N}} \log(G(t_i)) \qquad (17) \\ &= \sum_{i\in\mathcal{E}} \log\left(r^u(t_i)\right) - \log\left(1+\frac{1}{\kappa}\,H^u(t_i)\right) - \\ &\quad \sum_{i\in\mathcal{N}} \kappa\,\log\left(1+\frac{1}{\kappa}\,H^u(t_i)\right) \end{aligned}$$

Unfortunately, the standard method of episode splitting cannot be applied. Assume an episode $(0, t)$, starting at time zero, and ending at time t, and assume there is a covariate that changes its value at t_x $(0 < t_x < t)$. Then it would be quite possible to include this information into the calculation of the cumulated rate $H^u(t \mid x)$ which is needed for the calculation of the density and survivor function of the unconditional distribution. But the method of episode splitting only works when this calculation can be separated into two steps, the first step has information about the split $(0, t_x)$, and the second step has information about the split (t_x, t). But obviously, the unconditional density and survivor function can not be calculated in this two distinct steps. Therefore, all models described in the following subsections assume that covariates do not change during the given episodes. Of course, it would be possible to include information about time-dependent variables in the description of episodes in other ways; but this is currently not supported by TDA.

Secondly, one has to consider a situation with two or more transitions. In the same way as was shown in **6.17.1.2** for models without unobserved heterogeneity, it is possible to factor the total likelihood into transition-specific terms. Using the pseudo-survivor functions defined in (14), the

total likelihood can be written

$$\mathcal{L} = \prod_{j\in\mathcal{O}} \prod_{k\in\mathcal{D}_j} \prod_{i\in\mathcal{E}_{jk}} r_{jk}(t_i) \prod_{i\in\mathcal{N}_j} \tilde{G}_j(t_i)$$

Then, by inserting the expressions given in (16) and (14), and by taking the logarithm, the total log-likelihood becomes

$$\begin{aligned} \ell \;=\; & \sum_{j\in\mathcal{O}} \sum_{k\in\mathcal{D}_j} \sum_{i\in\mathcal{E}_{jk}} \log\left(r^u_{jk}(t_i)\right) - \log\left(1 + \frac{1}{\kappa}\, H^u_{jk}(t_i)\right) - \\ & \sum_{i\in\mathcal{N}_j} \kappa \log\left(1 + \frac{1}{\kappa_{jk}}\, H^u_{jk}(t_i)\right) \end{aligned} \tag{18}$$

As already mentioned, the models described in this paper are extensions of parametric transition rate models discussed in **6.17.3**. The underlying rate, $r^u_{jk}(t)$, always corresponds to one of those models. Therefore, as seen by (17) or (18), maximum likelihood estimation requires only little modifications.

In fact, the formulation of log-likelihoods and derivatives can use most of the formulas already given in **6.17.3**. For this reason, we will use the following notation. First we define new variables, d_{jk}, denoting the transition-specific variances of the gamma distributed stochastic terms in the models. These model parameters are analogously called the D-terms of the models. The possibility to link covariates to these model parameters is provided by exponential link functions, that is

$$d_{jk} = \frac{1}{k_{jk}} = \exp\left\{D^{(jk)}\,\delta^{(jk)}\right\}$$

$D^{(jk)}$ is a (row) vector of covariates with the first component always a constant one. $\delta^{(jk)}$ is the vector of associated coefficients to be estimated.

Secondly, to simplify the log-likelihood formulation, we define indicator variables

$$u_i \;=\; \begin{cases} -\frac{1}{d} - 1 & \text{if} \quad i \in \mathcal{E} \\ -\frac{1}{d} & \text{otherwise} \end{cases}$$

where $\mathcal{E}$ is the set of uncensored episodes. Furthermore, it is possible to omit indices for transitions since the total likelihood factors into transition-specific terms. The log-likelihood can be written, then, as

$$\ell = \sum_{i\in\mathcal{E}} \log\left(r^u(t_i)\right) + \sum_{i\in\mathcal{N}} u_i \log\left(1 + d\, H^u(t_i)\right)$$

with $r^u(t)$ and $H^u(t)$, respectively, the rate and the cumulative rate of the underlying model.

Finally, one can write the first and second derivatives of this log-likelihood in a way that can be used for all of the following models.

$$\begin{aligned}
\frac{\partial \ell}{\partial \alpha_{j_a}} &= \sum_{i\in\mathcal{E}} \frac{\partial \log(r^u(t_i))}{\partial \alpha_{j_a}} + \sum_{i\in\mathcal{N}} u_i \frac{d}{1+d\,H^u(t_i)} \frac{\partial H^u(t_i)}{\partial \alpha_{j_a}} \\
\frac{\partial \ell}{\partial \delta_{j_d}} &= \sum_{i\in\mathcal{N}} \left\{ \frac{\log(1+d\,H^u(t_i))}{d} + u_i \frac{d\,H^u(t_i)}{1+d\,H^u(t_i)} \right\} D_{i,j_d} \\
\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \alpha_{k_a}} &= \sum_{i\in\mathcal{E}} \frac{\partial^2 \log(r^u(t_i))}{\partial \alpha_{j_a} \partial \alpha_{k_a}} + \\
&\quad \sum_{i\in\mathcal{N}} u_i \frac{d}{1+d\,H^u(t_i)} \left\{ \frac{\partial^2 \log(H^u(t_i))}{\partial \alpha_{j_a} \partial \alpha_{k_a}} - \right. \\
&\qquad \left. \frac{d}{1+d\,H^u(t_i)} \frac{\partial H^u(t_i)}{\partial \alpha_{j_a}} \frac{\partial H^u(t_i)}{\partial \alpha_{k_a}} \right\} \\
\frac{\partial^2 \ell}{\partial \delta_{j_d} \partial \delta_{k_d}} &= \sum_{i\in\mathcal{N}} \left\{ \frac{2\,H^u(t_i)}{1+d\,H^u(t_i)} - \frac{\log(1+d\,H^u(t_i))}{d} + \right. \\
&\quad \left. u_i \frac{d\,H^u(t_i)}{1+d\,H^u(t_i)} \left[1 - \frac{d\,H^u(t_i)}{1+d\,H^u(t_i)} \right] \right\} D_{i,j_d} D_{i,k_d} \\
\frac{\partial^2 \ell}{\partial \alpha_{j_a} \partial \delta_{k_d}} &= \sum_{i\in\mathcal{N}} \frac{d}{1+d\,H^u(t_i)} \\
&\quad \left\{ \frac{1}{d} + u_i \left[1 - \frac{d\,H^u(t_i)}{1+d\,H^u(t_i)} \right] \right\} \frac{\partial H^u(t_i)}{\partial \alpha_{j_a}} D_{i,k_d}
\end{aligned}$$

These formulas, appropriate for a model with a single model term, can obviously be generalized for models with two or more terms. Consequently, maximum likelihood estimation of the models described in section **6.17.3**, with a stochastic term added, can use most of the expressions for the log-likelihood and its derivatives given there.

Exponential Models

This section describes the exponential model with a gamma-distributed stochastic term, v. The transition rate, now conditional on v, may be

written as

$$r(t \mid v) = a\,v \qquad a,\, v \geq 0$$

The underlying model, already described in 6.17.2.1, is the exponential, implying that

$$r^u(t) = a \qquad \text{and} \qquad H^u(t) = a\,t$$

The observed mixture distribution is described by

$$\begin{aligned} G(t) &= [1 + d\,a\,t]^{-\frac{1}{d}} \\ f(t) &= a\,[1 + d\,a\,t]^{-\frac{1}{d}-1} \\ r(t) &= a\,[1 + d\,a\,t]^{-1} \end{aligned}$$

d is the variance of the gamma distributed stochastic term. Obviously, if $d > 0$, there will be a negative time-dependence, although the underlying rate is a time-independent constant.

The model has two parameters. a, the constant transition rate of the underlying exponential model, and d, the variance of the mixing gamma distribution. In the TDA implementation of this model both parameters, the A- and D-term of the model, can be linked to covariates by an exponential link function. The model formulation for the transition rate from origin state j to destination state k is

$$\begin{aligned} r_{jk}(t) &= \frac{a_{jk}}{1 + d_{jk}\,a_{jk}\,t} \\ a_{jk} &= \exp\left(A^{(jk)}\alpha^{(jk)}\right) \\ d_{jk} &= \exp\left(D^{(jk)}\delta^{(jk)}\right) \end{aligned}$$

It is assumed that the first component of each of the covariate (row) vectors, $A^{(jk)}$ and $D^{(jk)}$, is a constant equal to one. The associated coefficient vectors, $\alpha^{(jk)}$ and $\delta^{(jk)}$, are the model parameters to be estimated.

The model can be requested by adding the `mix=1` parameter to the `rate` command for an exponential model. Maximum likelihood estimation is done following the outline given above. Using the notation introduced there, and dropping indices for transitions, the log-likelihood may be written as

$$\ell = \sum_{i \in \mathcal{E}} \log(a) + \sum_{i \in \mathcal{N}} u_i\,\log(1 + d\,a\,t_i)$$

The first and second derivatives are easily found following the remarks given above.

Weibull Models

This section describes the Weibull model with a gamma-distributed stochastic term, v. The transition rate, now conditional on v, is

$$r(t \mid v) = r^u(t)\, v \qquad v \geq 0$$

with an underlying rate and cumulative rate given by

$$r^u(t) = b\, a^b\, t^{b-1} \qquad \text{and} \qquad H^u(t) = (a\, t)^b$$

(see **6.17.3.3**). The observed mixture distribution is described by

$$\begin{aligned} G(t) &= \left[1 + d\,(a\, t)^b\right]^{-\frac{1}{d}} \\ f(t) &= b\, a^b\, t^{b-1}\, \left[1 + d\,(a\, t)^b\right]^{-\frac{1}{d}-1} \\ r(t) &= b\, a^b\, t^{b-1}\, \left[1 + d\,(a\, t)^b\right]^{-1} \end{aligned}$$

The model has three parameters. a, and b, are the parameters of the underlying Weibull distribution, and d represents the variance of the mixing gamma distribution. In the TDA implementation of this model all three parameters, the A-, B-, and D-term of the model, can be linked to covariates by an exponential link function. The model formulation for the transition rate from origin state j to destination state k is

$$\begin{aligned} r_{jk}(t) &= \frac{b_{jk}\, a_{jk}^{b_{jk}}\, t^{b_{jk}-1}}{1 + d_{jk}\,(a_{jk}\, t)^{b_{jk}}} \\ a_{jk} &= \exp\left(A^{(jk)}\alpha^{(jk)}\right) \\ b_{jk} &= \exp\left(B^{(jk)}\beta^{(jk)}\right) \\ d_{jk} &= \exp\left(D^{(jk)}\delta^{(jk)}\right) \end{aligned}$$

It is assumed that the first component of each of the covariate (row) vectors, $A^{(jk)}$, $B^{(jk)}$, and $D^{(jk)}$, is a constant equal to one. The associated coefficient vectors, $\alpha^{(jk)}$, $\beta^{(jk)}$, and $\delta^{(jk)}$, are the model parameters to be estimated.

The model can be requested by adding the `mix=1` parameter to the `rate` command for a Weibull model. Maximum likelihood estimation is done following the outline given in the introduction to this section. Using the notation introduced there, and dropping indices for transitions, the log-likelihood may be written as

$$\ell = \sum_{i \in \mathcal{E}} \log(b\, a^b\, t_i^{b-1}) + \sum_{i \in \mathcal{N}} u_i \, \log(1 + d\,(a\, t_i)^b)$$

First and second derivatives are easily found following the remarks given above, and using the formulas for derivatives given in **6.17.3.3**.

Sickle Models

This section describes the sickle model with a gamma-distributed stochastic term, v. The transition rate, now conditional on v, is

$$r(t \mid v) = r^u(t)\, v \qquad v \geq 0$$

with an underlying rate and cumulative rate defined by

$$\begin{aligned} r^u(t) &= a\, t \, \exp\left(-\frac{t}{b}\right) \\ H^u(t) &= a\, b \left[b - (t+b)\, \exp\left(-\frac{t}{b}\right)\right] \end{aligned}$$

(see **6.17.3.4**). The observed mixture distribution is described by

$$\begin{aligned} G(t) &= \left[1 + d\, a\, b \left\{b - (t+b)\, \exp\left(-\frac{t}{b}\right)\right\}\right]^{-\frac{1}{d}} \\ f(t) &= a\, t \, \exp\left(-\frac{t}{b}\right) \left[1 + d\, a\, b \left\{b - (t+b)\, \exp\left(-\frac{t}{b}\right)\right\}\right]^{-\frac{1}{d}-1} \\ r(t) &= a\, t \, \exp\left(-\frac{t}{b}\right) \left[1 + d\, a\, b \left\{b - (t+b)\, \exp\left(-\frac{t}{b}\right)\right\}\right]^{-1} \end{aligned}$$

The model has three parameters. a, and b, are the parameters of the underlying sickle model, and d represents the variance of the mixing gamma distribution. In the TDA implementation of this model all three parameters, the A-, B-, and D-term of the model, can be linked to covariates by an exponential link function. The model formulation for the

transition rate from origin state j to destination state k is

$$\begin{aligned} r_{jk}(t) &= a_{jk}\, t\, \exp\left(-\frac{t}{b_{jk}}\right) \\ &\quad \left[1 + d_{jk}\, a_{jk}\, b_{jk}\, \left\{b_{jk} - (t + b_{jk})\, \exp\left(-\frac{t}{b_{jk}}\right)\right\}\right]^{-1} \\ a_{jk} &= \exp\left(A^{(jk)}\alpha^{(jk)}\right) \\ b_{jk} &= \exp\left(B^{(jk)}\beta^{(jk)}\right) \\ d_{jk} &= \exp\left(D^{(jk)}\delta^{(jk)}\right) \end{aligned}$$

It is assumed that the first component of each of the covariate (row) vectors, $A^{(jk)}$, $B^{(jk)}$, and $D^{(jk)}$, is a constant equal to one. The associated coefficient vectors, $\alpha^{(jk)}$, $\beta^{(jk)}$, and $\delta^{(jk)}$, are the model parameters to be estimated.

The model can be requested by adding the `mix=1` parameter to the `rate` command for a sickle model. Maximum likelihood estimation is done following the outline given in the introduction. Using the notation introduced there, and dropping indices for transitions, the log-likelihood may be written as

$$\ell = \sum_{i \in \mathcal{E}} \log(b\, a^b\, t_i^{b-1}) + \sum_{i \in \mathcal{N}} u_i\, \log(1 + d\,(a\, t_i)^b)$$

First and second derivatives are easily found following the remarks given above, and using the formulas for derivatives given in **6.17.3.4**.

Log-Logistic Models

This section describes the log-logistic model with a gamma-distributed stochastic term, v. The transition rate, now conditional on v, is

$$r(t \mid v) = r^u(t)\, v \qquad v \geq 0$$

with an underlying rate and cumulative rate defined by

$$r^u(t) = \frac{b\, a^b\, t^{b-1}}{1 + (a\, t)^b} \qquad \text{and} \qquad H^u(t) = \log\left(1 + (a\, t)^b\right)$$

(see 6.17.3.5). The observed mixture distribution is described by

$$\begin{aligned} G(t) &= \left[1 + d\,\log(1 + (a\,t)^b)\right]^{-\frac{1}{d}} \\ f(t) &= \frac{b\,a^b\,t^{b-1}}{1 + (a\,t)^b}\left[1 + d\,\log(1 + (a\,t)^b)\right]^{-\frac{1}{d}-1} \\ r(t) &= \frac{b\,a^b\,t^{b-1}}{1 + (a\,t)^b}\left[1 + d\,\log(1 + (a\,t)^b)\right]^{-1} \end{aligned}$$

The model has three parameters. a, and b, are the parameters of the underlying log-logistic distribution, and d represents the variance of the mixing gamma distribution. In the TDA implementation of this model all three parameters, the A-, B-, and D-term of the model, can be linked to covariates by an exponential link function. The model formulation for the transition rate from origin state j to destination state k is

$$\begin{aligned} r_{jk}(t) &= \frac{b_{jk}\,a_{jk}^{b_{jk}}\,t^{b_{jk}.1}}{1 + (a_{jk}\,t)^{b_{jk}}}\left[1 + d_{jk}\,\log(1 + (a_{jk}\,t)^{b_{jk}})\right]^{-1} \\ a_{jk} &= \exp\left(A^{(jk)}\alpha^{(jk)}\right) \\ b_{jk} &= \exp\left(B^{(jk)}\beta^{(jk)}\right) \\ d_{jk} &= \exp\left(D^{(jk)}\delta^{(jk)}\right) \end{aligned}$$

It is assumed that the first component of each of the covariate (row) vectors, $A^{(jk)}$, $B^{(jk)}$, and $D^{(jk)}$, is a constant equal to one. The associated coefficient vectors, $\alpha^{(jk)}$, $\beta^{(jk)}$, and $\delta^{(jk)}$, are the model parameters to be estimated.

The model can be requested by adding the `mix=1` parameter to the `rate` command for a log-logistic model. Maximum likelihood estimation is done following the outline given in the introduction. Using the notation introduced there, and dropping indices for transitions, the log-likelihood may be written as

$$\ell = \sum_{i\in\mathcal{E}} \log(b\,a^b\,t_i^{b-1}) + \sum_{i\in\mathcal{N}} u_i\,\log(1 + d\,(a\,t_i)^b)$$

First and second derivatives are easily found following the remarks given above, and using the formulas for derivatives given in 6.17.3.5.

Log-Normal Models

This section describes the log-normal model with a gamma-distributed stochastic term, v. The transition rate, now conditional on v, is

$$r(t \mid v) = r^u(t)\, v \qquad v \geq 0$$

with an underlying rate and cumulative rate defined by

$$\begin{aligned} r^u(t) &= \frac{1}{b\,t}\,\frac{\phi\left(z_t\right)}{1-\Phi\left(z_t\right)} \qquad z_t = \frac{\log(t)-a}{b} \\ H^u(t) &= -\log\left(1-\Phi\left(z_t\right)\right) \end{aligned}$$

(see 6.17.3.6). The observed mixture is described by

$$\begin{aligned} G(t) &= \left[1-d\,\log\left\{1-\Phi\left(z_t\right)\right\}\right]^{-\frac{1}{d}} \\ f(t) &= \frac{1}{b\,t}\,\frac{\phi\left(z_t\right)}{1-\Phi\left(z_t\right)}\left[1-d\,\log\left\{1-\Phi\left(z_t\right)\right\}\right]^{-\frac{1}{d}-1} \\ r(t) &= \frac{1}{b\,t}\,\frac{\phi\left(z_t\right)}{1-\Phi\left(z_t\right)}\left[1-d\,\log\left\{1-\Phi\left(z_t\right)\right\}\right]^{-1} \end{aligned}$$

The model has three parameters. a, and b, are the parameters of the underlying log-normal model, and d represents the variance of the mixing gamma distribution. In the TDA implementation of this model all three parameters, the A-, B-, and D-term of the model, can be linked to covariates by an exponential link function. The model formulation for the transition rate from origin state j to destination state k is

$$r_{jk}(t) = \frac{1}{b_{jk}\,t}\,\frac{\phi\left(z_{jk}(t)\right)}{1-\Phi\left(z_{jk}(t)\right)}\left[1-d_{jk}\,\log\left\{1-\Phi\left(z_{jk}(t)\right)\right\}\right]^{-1}$$

with

$$\begin{aligned} z_{jk}(t) &= \frac{\log(t)-a_{jk}}{b_{jk}} \\ a_{jk} &= \exp\left(A^{(jk)}\alpha^{(jk)}\right) \\ b_{jk} &= \exp\left(B^{(jk)}\beta^{(jk)}\right) \\ d_{jk} &= \exp\left(D^{(jk)}\delta^{(jk)}\right) \end{aligned}$$

It is assumed that the first component of each of the covariate (row) vectors, $A^{(jk)}$, $B^{(jk)}$, and $D^{(jk)}$, is a constant equal to one. The associated coefficient vectors, $\alpha^{(jk)}$, $\beta^{(jk)}$, and $\delta^{(jk)}$, are the model parameters to be estimated.

The model can be requested by adding the `mix=1` parameter to the `rate` command for an log-normal model. Maximum likelihood estimation is done following the outline given in the introduction. Using the notation introduced there, and dropping indices for transitions, the log-likelihood may be written as

$$\ell = \sum_{i \in \mathcal{E}} \log \left(\frac{1}{b\,t} \, \frac{\phi\left(z_t\right)}{1 - \Phi\left(z_t\right)} \right) + \sum_{i \in \mathcal{N}} u_i \, \log \left\{ 1 - d \, \log \left(1 - \Phi\left(z_t\right) \right) \right\}$$

First and second derivatives are easily found following the remarks given above, and using the formulas for derivatives given in **6.17.3.6**.

6.17.5 User-defined Rate Models

This chapter describes the `frml` command that can be used to estimate transition rate models based on a user-defined likelihood. In addition, the chapter discusses some examples. We thank Francesco Billari who developed these applications of the `frml` command. The sections are:

6.17.5.1 The `frml` Command

The command for estimating user-defined rate models is

```
frml (parameters) = log-likelihood function;
```

This command is basically identical with the `fml` command described in 6.11.2. The main difference is that the `frml` command requires an episode data structure, and the log-likelihood function is summed over all episodes (or episode splits) which are currently defined; furthermore, one can use the episode data operators (`org`, `des`, `ts`, `tf`) and type 5 variables (defined inside the `edef` command) in the definition of the log-likelihood function.

The right-hand side of the command must provide the log-likelihood function for a transition rate model. All other parameters (on the left-hand side) are optional and can be used to select an algorithm for function minimization, to define starting values, to change the print format, and so on. In principle, one can use all parameters to control maximum likelihood estimation which have been explained in 5.6. It should also be possible to use the `con` parameter to define equality constraints.

All minimization algorithms discussed in 5.6.2 can be used; default is the Newton I algorithm (`mina=5`). If required, Wengert's method is used to calculate first and second derivatives of the log-likelihood function.

The remainder of this section provides some elementary examples to illustrate the `frml` command. Additional discussion and examples will be given in subsequent sections. All examples will use the episode data set `rrdat.1` described in 3.3.3.

Example 1 Box 1 shows the command file `frml1.cf` that can be used to replicate the estimation of a simple exponential model (see the example in 6.17.2.1). The log-likelihood function is

$$\ell = \sum_{i \in \mathcal{E}} \log(r(t_i)) + \sum_{i \in \mathcal{N}} \log(G(t_i))$$

In our example, the rate does not depend on time,

$$r(t_i) = r = \exp(\alpha_0 + \texttt{COHO2}\alpha_1 + \texttt{COHO3}\alpha_2 + \texttt{W}\alpha_3)$$

Box 1 Command file `frml1.cf` (exponential model)

```
nvar(
  dfile = rrdat.1,      # data file
  ID    [3.0] = c1,     # identification number
  SN    [2.0] = c2,     # spell number
  TS    [3.0] = c3,     # starting time
  TF    [3.0] = c4,     # ending time
  SEX   [2.0] = c5,     # sex (1 men, 2 women)
  TI    [3.0] = c6,     # interview date
  TB    [3.0] = c7,     # birth date
  TE    [3.0] = c8,     # entry into labor market
  TMAR  [3.0] = c9,     # marriage date (0 if no marriage)
  PRES  [3.0] = c10,    # prestige of current job
  PRESN [3.0] = c11,    # prestige of next job
  EDU   [2.0] = c12,    # highest educational attainment

  # define additional variables

  COHO1 = ge(TB,348) & le(TB,384),  # birth cohort 1
  COHO2 = ge(TB,468) & le(TB,504),  # birth cohort 2
  COHO3 = ge(TB,588) & le(TB,624),  # birth cohort 3
  W     = SEX[2],                   # women = 1

  DES [1.0] = if eq(TF,TI) then 0 else 1,
  DUR [3.0] = TF - TS + 1,
);
edef(          # define single episode data
  ts = 0,      # starting time
  tf = DUR,    # ending time
  org = 0,     # origin state
  des = DES,   # destination state
);
frml (
  xp = -4,0,0,0,      # starting values

) = rate = exp(a0 + COHO2 * a1 + COHO3 * a2 + W * a3),
    l1   = if(DES,log(rate),0),
    fn   = l1 - rate * DUR;
```

and the survivor function is simply

$$G(t_i) = \exp(-r\, t_i)$$

yielding the log-likelihood function

$$\ell = \sum_{i \in \mathcal{E}} \log(r) - \sum_{i \in \mathcal{N}} r t_i$$

Box 2 Part of standard output from `frml1.cf`

```
ML estimation of user-defined rate model.

Function definition:
rate     = exp(a0+COHO2*a1+COHO3*a2+W*a3)
l1       = if(DES,log(rate),0)
fn       = l1-rate*DUR

Function will be summed over 600 data matrix cases.
Using episode data.

Maximum likelihood estimation.
Algorithm 5: Newton (I)

Number of model parameters: 4
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Got 4 starting value(s) from xp parameter.

Idx  Parameter   Starting value
  1  a0          -4.00000000e+00
  2  a1           0.00000000e+00
  3  a2           0.00000000e+00
  4  a3           0.00000000e+00

Convergence reached in 6 iterations.
Number of function evaluations: 7 (7,7)

Maximum of log likelihood: -2475.44
Norm of final gradient vector: 1.10499e-11
Last absolute change of function value: 3.30667e-15
Last relative change in parameters: 5.84648e-07

Idx  Parameter       Value      Error    Value/E  Signif
------------------------------------------------------------
  1  a0            -5.0114     0.0843   -59.4446  1.0000
  2  a1             0.5341     0.1120     4.7686  1.0000
  3  a2             0.6738     0.1152     5.8472  1.0000
  4  a3             0.5065     0.0942     5.3746  1.0000

Log likelihood (starting values): -2578.9484
Log likelihood (final estimates): -2475.4383
```

Box 3 Part of Command file `frml1m.cf`

```
frml (
    xp = -4,0,0,0,      # starting values
         -4,0,0,0,
         -4,0,0,0,

) = rate1 = exp(a10 + COHO2 * a11 + COHO3 * a12 + W * a13),
    rate2 = exp(a20 + COHO2 * a21 + COHO3 * a22 + W * a23),
    rate3 = exp(a30 + COHO2 * a31 + COHO3 * a32 + W * a33),
    rate  = rate1 + rate2 + rate3,
    l1    = if(eq(DES,1),log(rate1),
            if(eq(DES,2),log(rate2),
            if(eq(DES,3),log(rate3),0))),
    fn    = l1 - rate * DUR;
```

This expression is used in the command file `frml1.cf`. Part of the standard output is shown in Box 2. The estimated model parameters and standard errors are identical with the results shown in **6.17.2.1**.

Example 2 To replicate the estimation of an exponential model with three alternative destination states (see **6.17.2.1**), one can use the `frml` command shown in Box 3. This command is part of command file `frml1m.cf` (not shown) which is basically identical to command file `rt1m.cf` to set up an episode data structure for three alternative destination states.

Example 3 The `frml` command in Box 4 illustrate the definition of a log-likelihood function for a piecewise constant exponential model with 9 time periods. The specification is identical with the example discussed in **6.17.2.2** and should provide identical results. Of course, there are many different possibilities to define this log-likelihood function.

This example can also be used to illustrate equality constraints. If adding the parameters

$$\text{con} \;=\; b_i - b_{i+1} = 0, \qquad (i = 1, \ldots, 8)$$

to the `frml` command in `frml2.cf`, one should get the estimation results for a simple exponential model (see command file `frml2c.cf` in the example archive).

Example 4 The `frml` command in Box 5 estimates a Weibull model. It is a replication of the example discussed in **6.17.3.3**. The log-likelihood can easily be modified to provide alternative specifications of the Weibull model. For example, removing the line `bb=exp(b0)`, would result in a

Box 4 Part of Command file `frml2.cf`

```
frml (
    xp = -4,-4,-4,-4,-4,-4,-4,-4,-4,0,0,0,     # starting values

) = bb = COHO2 * b1 + COHO3 * b2 + W * b3,
    p1 = ge(tf, 0) & lt(tf,12),
    p2 = ge(tf,12) & lt(tf,24),
    p3 = ge(tf,24) & lt(tf,36),
    p4 = ge(tf,36) & lt(tf,48),
    p5 = ge(tf,48) & lt(tf,60),
    p6 = ge(tf,60) & lt(tf,72),
    p7 = ge(tf,72) & lt(tf,84),
    p8 = ge(tf,84) & lt(tf,96),
    p9 = ge(tf,96),
    rr = if(p1,a1,
         if(p2,a2,
         if(p3,a3,
         if(p4,a4,
         if(p5,a5,
         if(p6,a6,
         if(p7,a7,
         if(p8,a8,
         if(p9,a9,0)))))))),

    s1 = 12 * exp(a1 + bb),
    s2 = s1 + 12 * exp(a2 + bb),
    s3 = s2 + 12 * exp(a3 + bb),
    s4 = s3 + 12 * exp(a4 + bb),
    s5 = s4 + 12 * exp(a5 + bb),
    s6 = s5 + 12 * exp(a6 + bb),
    s7 = s6 + 12 * exp(a7 + bb),
    s8 = s7 + 12 * exp(a8 + bb),

    lsurv = if(p1,tf * exp(a1 + bb),
            if(p2,s1 + (tf - 12) * exp(a2 + bb),
            if(p3,s2 + (tf - 24) * exp(a3 + bb),
            if(p4,s3 + (tf - 36) * exp(a4 + bb),
            if(p5,s4 + (tf - 48) * exp(a5 + bb),
            if(p6,s5 + (tf - 60) * exp(a6 + bb),
            if(p7,s6 + (tf - 72) * exp(a7 + bb),
            if(p8,s7 + (tf - 84) * exp(a8 + bb),
            if(p9,s8 + (tf - 96) * exp(a9 + bb),0)))))))),

    l1   = if(DES,rr + bb,0),
    fn   = l1 - lsurv;
```

Box 5 Part of Command file `frml3.cf` (Weibull model)

```
frml (
    xp = -4,0,0,0,0,

) = aa = exp( a0 + COHO2 * a1 + COHO3 * a2 + W * a3 ),
    bb = exp( b0 ),
    l1 = if(DES,log(bb * aa^bb * tf^(bb - 1)),0),
    fn = l1 - (aa * tf)^bb;
```

Box 6 Part of Command file `frml4.cf` (log-normal model)

```
frml (
    xp = 4,0,0,0,0,

) = aa = a0 + COHO2 * a1 + COHO3 * a2 + W * a3,
    bb = exp( b0 ),
    zz = (log(tf) - aa) / bb,
    qz = 1 - nd(zz),
    vz = ndf(zz) / qz,
    l1   = if(DES,log(vz / (bb * tf)),0),
    fn   = l1 + log(qz);
```

direct estimate of `bb` and its standard error. (One should change, then, the starting value for this parameter into a 1.)

Example 5 Box 6 shows an `frml` command to estimate a log-normal model. It is a replication of the example discussed in **6.17.3.6**. The definition of the log-likelihood function uses the operators `nd` and `ndf` for the standard normal distribution and density function, respectively. This is possible since these operators allow for automatic differentiation.

Example 6 To replicate the estimation of a generalized gamma model (see **6.17.3.7**), we use command file `frml5.cf`. Part of this command file is shown in Box 7. As discussed in **6.17.3.7**, the log-likelihood requires the gamma function and the incomplete gamma integral. Since TDA automatically provides derivatives for the logarithm of the gamma function (`lgam` operator) and for the incomplete gamma integral (`icg` operator), the log-likelihood can be directly written, without numerical integration. See Box 7. The resulting parameter estimates are identical to those achieved with the `rate` command in **6.17.3.7**. Note that we have used the BFGS algorithm (`mina=4`), since the default Newton algorithm would need better starting values. However, the final covariance matrix

Box 7 Part of Command file `frml5.cf` (gamma model)

```
frml (
    mina = 4,
    xp = 5,0,0,0,0,

) = kk = 1,
    aa = a0 + COHO2 * a1 + COHO3 * a2 + W * a3,
    bb = exp(b0),
    zz = (log(tf) - aa) / bb,
    qt = kk * exp(zz / sqrt(kk)),
    rr = (kk - 0.5) * log(kk) + (sqrt(kk) * zz - qt) -
                                    (b0 + log(tf) + lgam(kk)),
    icc = icg(qt,kk),
    fn = if (DES,rr,log(1 - icc));
```

is based on (automatically calculated) second derivatives.

Episode Splitting. The `frml` command expects the definition of a log-likelihood function on the right-hand side. In general, this will be a function of data matrix variables (including possibly type 5 variables) and model parameters. This function is summed over all episodes, or episode splits, which are available in the currently defined episode data structure. Consequently, if the episode data structure contains episode splits (given by the input data or by using the `split` option in the `edef` command), summation is over episode splits. It is easy, therefore, to apply the method of episode splitting to user-defined transition rate models.

Example 7 To illustrate, we use the simple exponential model (Example 1). The new command file is `frml1s.cf`, shown in Box 8. Purely for illustration, we define a variable `SV=50` to be used in the `edef` command for episode splitting. Every episode which is longer than 50 is then split at $t = 50$, resulting in 845 episode splits.

To get the same results as in Example 1, the definition of the log-likelihood must be modified. Instead of using the duration (previously `DUR`), we have to use starting and ending times of the splits. This information is available by the `ts` and `tf` operators. Also, instead of using the exit status of the episodes (previously given by `DES`), we now have to use the exit status of the episode splits; available by the operator `des`. Using this modified set up of the log-likelihood function, we get identical results as seen in Box 9.

Box 8 Command file `frml1s.cf`

```
nvar(
    dfile = rrdat.1,      # data file
    ID    [3.0] = c1,     # identification number
    SN    [2.0] = c2,     # spell number
    TS    [3.0] = c3,     # starting time
    TF    [3.0] = c4,     # ending time
    SEX   [2.0] = c5,     # sex (1 men, 2 women)
    TI    [3.0] = c6,     # interview date
    TB    [3.0] = c7,     # birth date
    TE    [3.0] = c8,     # entry into labor market
    TMAR  [3.0] = c9,     # marriage date (0 if no marriage)
    PRES  [3.0] = c10,    # prestige of current job
    PRESN [3.0] = c11,    # prestige of next job
    EDU   [2.0] = c12,    # highest educational attainment
    COHO1 = ge(TB,348) & le(TB,384),  # birth cohort 1
    COHO2 = ge(TB,468) & le(TB,504),  # birth cohort 2
    COHO3 = ge(TB,588) & le(TB,624),  # birth cohort 3
    W     = SEX[2],                   # women = 1
    DES [1.0] = if eq(TF,TI) then 0 else 1,
    DUR [3.0] = TF - TS + 1,
    SV = 50,
);
edef(           # define single episode data
    ts = 0,     # starting time
    tf = DUR,   # ending time
    org = 0,    # origin state
    des = DES,  # destination state
    split = SV, # episode splitting at t = 50
);
frml (
    xp = -4,0,0,0,     # starting values

) = rate = exp(a0 + COHO2 * a1 + COHO3 * a2 + W * a3),
    l1   = if(des,log(rate),0),
    fn   = l1 - rate * (tf - ts);
```

Box 9 Part of standard output from command file `frml1s.cf`

```
ML estimation of user-defined rate model.

Function definition:
rate    = exp(a0+COHO2*a1+COHO3*a2+W*a3)
l1      = if(des,log(rate),0)
fn      = l1-rate*(tf-ts)

Function will be summed over 600 data matrix cases.
Using episode data (845 episode splits).

Convergence reached in 6 iterations.
Number of function evaluations: 7 (7,7)

Maximum of log likelihood: -2475.44
Norm of final gradient vector: 1.08916e-11
Last absolute change of function value: 1.83704e-16
Last relative change in parameters: 5.84648e-07

Idx  Parameter       Value      Error    Value/E  Signif
-----------------------------------------------------------
  1  a0            -5.0114     0.0843   -59.4446  1.0000
  2  a1             0.5341     0.1120     4.7686  1.0000
  3  a2             0.6738     0.1152     5.8472  1.0000
  4  a3             0.5065     0.0942     5.3746  1.0000

Log likelihood (starting values): -2578.9484
Log likelihood (final estimates): -2475.4383
```

6.17.5.2 The Hernes Model

To illustrate estimation of the Hernes model (originally proposed by Hernes [1972], see also Diekmann [1990], Wu [1990]), we follow the model formulation given by Wu [1990]. The (defective) distribution function is given by

$$F(t) = \frac{1}{1 + \sigma \exp(\beta\lambda^t)}$$

with parameters $\sigma > 0$, $\beta > 0$, and $0 < \lambda < 1$. The corresponding survivor function is

$$G(t) = \frac{1}{1 + \exp(-\beta\lambda^t)/\sigma}$$

and the density function is

$$f(t) = \frac{-\sigma \exp(\beta\lambda^t)\beta\lambda^t \log(\lambda)}{(1 + \sigma \exp(\beta\lambda^t))^2}$$

The log-likelihood for model estimation can be written

$$\ell = \sum_{i \in \mathcal{E}} \log(f(t_i; \sigma, \beta, \lambda) + \sum_{i \in \mathcal{Z}} \log(G(t_i; \sigma, \beta, \lambda) \tag{1}$$

In general, one can link covariates to all three model parameters. It seems reasonable to use the following link functions:

$$\begin{aligned} \beta &= \exp(b) \\ \sigma &= \exp(s) \\ \lambda &= \exp(l)/(1 + \exp(l)) \end{aligned} \tag{2}$$

Example 1 To illustrate, we begin with a model formulation without covariates. A suitable command file to be used with our standard episode data set is `frml8.cf`. Part of this command file (the `frml` command) is shown in Box 1. The log-likelihood formulation follows (1) and uses the link functions proposed in (2). Part of the standard output is shown in Box 2. Some numerical problems occurred since we have not specified suitable starting values.

Box 1 Part of command file `frml8.cf`

```
frml (
    mina = 4,
) = sigma  = exp(s),
    lambda = exp(l)/(1 + exp(l)),
    beta   = exp(b),
    ff     = sigma * exp(beta * lambda^tf),
    ff1    = ff + 1,
    dens   = -(ff * beta * lambda^tf * log(lambda)) / (ff1 * ff1),
    surv   = 1 - 1 / ff1,
    fn     = if(des,log(dens),log(surv));
```

Example 2 The TDA example archive contains another command file, `frml8a.cf`, that can be used to estimate a Hernes model with covariates linked to the β parameter of the model. To achieve convergence, we have used the parameter estimates from Example 1 as starting values.

Box 2 Part of standard output from command file `frml8.cf`

```
ML estimation of user-defined rate model.

Function definition:
sigma   = exp(s)
lambda  = exp(l)/(1+exp(l))
beta    = exp(b)
ff      = sigma*exp(beta*lambda^tf)
ff1     = ff+1
dens    = -(ff*beta*lambda^tf*log(lambda))/(ff1*ff1)
surv    = 1-1/ff1
fn      = if(des,log(dens),log(surv))

Maximum likelihood estimation.
Algorithm 5: Newton (I)

Number of model parameters: 3
Type of covariance matrix: 2

Idx  Parameter   Starting value
  1  b            0.00000000e+00
  2  l            0.00000000e+00
  3  s            0.00000000e+00

Convergence reached in 10 iterations.
Number of function evaluations: 25 (25,25)

Numerical problems.
  1 : overflow/underflow in exp() or log().
  3 : log-likelihood positive.

Idx  Parameter       Value      Error    Value/E  Signif
-------------------------------------------------------
  1  b              1.5856     0.0378    41.9814  1.0000
  2  l              3.8236     0.0599    63.8607  1.0000
  3  s             -1.8379     0.1305   -14.0821  1.0000

Log likelihood (starting values): -16552.9633
Log likelihood (final estimates): -2526.3743
```

6.17.5.3 The Coale-McNeil Model

Another model sometimes used in demographical research is the Coale-McNeil model, originally proposed by Coale and McNeil [1972]. For a discussion of the model, see Huinink and Henz [1993], Wu [1990]. The density function is

$$f(t) = \sigma\lambda \exp\{-\beta(t-\mu) - \exp(-\lambda(t-\mu)) - \log(\Gamma(\beta/\lambda))\}$$

with parameters $\mu > 0$, $\beta > 0$, and $\lambda > 0$. Slightly more general, one can also consider the defective density

$$f_\sigma(t) = \sigma\, f(t) \qquad 0 \le \sigma \le 1$$

To derive the distribution function

$$F_\sigma(t) = \int_{-\infty}^{t} f_\sigma(\tau)\,\mathrm{d}\,\tau$$

one can apply the substitution $\tau = \mu - \log(u)/\lambda$, to get

$$F_\sigma(t) = \sigma\,(1 - I(\exp(-\lambda(t-\mu)), \beta/\lambda))$$

where I denotes the incomplete gamma integral. To estimate this model, one often assumes that β/λ is constant. As proposed by Rodriguez and Trussell, $\beta/\lambda = 0.604$ (see Wu [1990], p. 192). It seems reasonable, then, to use the parameterization

$$\begin{aligned} \beta/\lambda &= \text{constant} \\ \lambda &= \exp(l) \\ \mu &= \exp(m) \\ \sigma &= \exp(s)/(1+\exp(s)) \end{aligned}$$

Example 1 To illustrate, we begin with $\beta/\lambda = 0.604$ and $\sigma = 1$, and use our standard episode data set. The `frml` command for model estimation (part of command file `frml9.cf`) is shown in Box 1. The log-likelihood uses TDA's operators for the logarithm of the gamma function (`lgam`) and the incomplete gamma integral (`icg`). We use the BFGS algorithm (`mina=4`) since the default Newton algorithm is very sensitive

Box 1 Part of command file `frml9.cf`

```
frml (
    mina = 4,
    xp = -3,2,

 )= bg     = 0.604,
    sigma  = exp(s) / (1 + exp(s)),
    lambda = exp(l),
    mue    = exp(m),
    beta   = bg * lambda,
    tl     = exp(-lambda * (tf - mue)),
    ldens  = log(sigma) + l - beta * (tf - mue) - tl - lgam(bg),
    surv   = 1 - sigma * (1 - icg(tl,bg)),
    fn     = if (DES,ldens,log(surv));
```

to starting values. (In fact, the BFGS algorithm could also be used with default zero starting values.) Estimation results are shown in Box 2.

It should also be possible to link covariates to the model parameters. As an illustration, the TDA example archive contains the command file `frml9a.cf` where our standard covariates (`COHO2`, `COHO3`, and `W`) are linked to the λ parameter. This results in a significantly better fit, the log-likelihood is -2322.96.

Box 2 Part of standard output from command file `frml9.cf`

```
ML estimation of user-defined rate model.

Function definition:
bg       = 0.604
sigma    = exp(s)/(1+exp(s))
lambda   = exp(l)
mue      = exp(m)
beta     = bg*lambda
tl       = exp(-lambda*(tf-mue))
ldens    = log(sigma)+l-beta*(tf-mue)-tl-lgam(bg)
surv     = 1-sigma*(1-icg(tl,bg))
fn       = if(DES,ldens,log(surv))

Function evaluation: sum over 600 data matrix cases.
Using episode data.

Maximum likelihood estimation.
Algorithm 4: BFGS

Idx  Parameter   Starting value
  1  l           -3.00000000e+00
  2  m            2.00000000e+00
  3  s            0.00000000e+00

Convergence reached in 13 iterations.
Number of function evaluations: 22 (22,1)

Maximum of log likelihood: -2529.59
Norm of final gradient vector: 8.34764e-07

Idx  Parameter       Value      Error    Value/E  Signif
-------------------------------------------------------
  1  l            -3.0653     0.0454   -67.4629  1.0000
  2  m             2.8902     0.0685    42.1764  1.0000
  3  s             1.8505     0.1421    13.0217  1.0000

Log likelihood (starting values): -2699.6069
Log likelihood (final estimates): -2529.5933
```

6.17.5.4 Models with Several Domains

In unpublished papers, Francesco Billari has proposed to consider transition rate models with two (or more) domains. In the most simple case, the modeling approach is

$$r(t) = \begin{cases} r_a(t) & \text{if } t \le t_a \\ r_a(t) + r_b(t) & \text{if } t > t_a \end{cases}$$

where $r_a(t)$ is the rate for the first domain, $[0, t_a]$, and $r_a(t) + r_b(t)$ is the rate for the second domain, (t_a, ∞). A possible application, proposed by Billari, is to investigate age at first marriage. Until a certain age t_a, a very low constant rate $r_a(t)$ is assumed, then, beginning at age t_a, individuals enter the "'essential"' risk period for becoming married and the rate is given by $r_a(t) + r_b(t)$.

In principle, it should be possible to regard t_a as a model parameter that can be estimated. The problem is, of course, that the log-likelihood is not continuously differentiable in t_a. As a possible strategy, one can try an algorithm that does not require derivatives. Another strategy would be to estimate the model with a series of differently fixed values of t_a. Of course, the best strategy would be to apply an algorithm for non-smooth function optimization; however, this is not yet available in TDA.

Since this modeling approach is not well suited for our standard example data set, we do not give an illustration. However, the TDA example archive contains the command file `frml10.cf` which can be used for some experiments. It uses a simple exponential model for $r_a(t)$ and a log-normal model for $r_b(t)$.

6.17.6 Discrete Time Rate Models

This chapter discusses discrete time transition rate models. Basic references are Allison [1982] and Hamerle and Tutz [1989]. The sections are:

6.17.6.1 Introduction

Discrete time models begin with the assumption that transitions from a given origin state to a new destination state can only occur at some discrete points in time. It is possible then to represent the time axis by the natural numbers $(1, 2, 3, \ldots)$. To describe episodes in a discrete time framework we can use the same notation as introduced in **3.3.1** for a continuous time framework:

$$(o_i, d_i, t_i^s, t_i^f, x_i(t)) \qquad i = 1, \ldots, N$$

t_i^s is the starting time of the episode, t_i^f is the ending time. o_i is the origin state of the episode, and d_i is the destination state. $x_i(t)$ is a vector of covariates, possibly time-varying, connected with the episode.

The only important difference is in the interpretation of starting and ending times. We first concentrate on ending times and assume that all starting times are zero. As already mentioned, values of ending times are natural numbers indexing the discrete points in time when events can take place. Consequently, the duration of episodes is represented by a discrete random variable T^*. (To distinguish discrete time concepts from the corresponding continuous time concepts we mark all symbols by an asterisk.) For all practical purposes we can restrict T^* to be finite with possible values $1, \ldots, q$.

$T_i^* = t_i$ means that in the ith episode an event takes place *at* t_i, or the episode is censored *at* this point in time. To understand this, it may be helpful to imagine an underlying continuous time axis with intervals defined by

$$\tau_0 < \tau_1 < \tau_2 < \ldots < \tau_q \qquad \text{with} \quad \tau_0 = 0$$

A continuous variable T, defined on this time axis, may then be used to define a one-to-one relation

$$T^* = t \;\Leftrightarrow\; T \in [\tau_{t-1}, \tau_t) \qquad t = 1, \ldots, q$$

That an event takes place *at* the discrete point in time t may be interpreted, then, as an event taking place in the tth interval on the underlying continuous time axis. The meaning of a right censored episode *at*

the discrete point in time t is, accordingly, that no event has happened up to the end of the tth interval.

With these guidelines for interpretation, we can define some concepts for a statistical description; first assuming a single possible event. We may start with the discrete analogon to the density function for a continuous time variable which is the probability function for the discrete time variable T^*. It is simply defined as

$$f^*(t) = \Pr(T^* = t) \qquad t = 1, \ldots, q$$

with

$$\sum_{t=1}^{q} f^*(t) = 1$$

Next we can define a discrete analogon for the survivor function. To be consistent with the interpretation of right censored episodes, the definition is

$$G^*(t) = \Pr(T^* > t) \tag{1}$$

meaning that at least at t the individual is still in the origin state. Finally, a discrete analogon for the transition rate can be defined as

$$r^*(t) = \Pr(T^* = t \mid T^* \geq t)$$

Obviously, this is a conditional probability, meaning that the episode ended with an event at t given that no event occurred before t. This implies that

$$0 \leq r^*(t) \leq 1$$

what is different from the continuous time case. In that case the transition rate can assume values greater than one; now the rate is restricted to the range zero to one, and of course, the formulation of models must be consistent with this fact.

Another difference to the continuous time case is in the expression of the transition rate by the density and survivor functions. As a consequence of the definition in (1), the formulation is

$$r^*(t) = \frac{f^*(t)}{G^*(t-1)} = \frac{f^*(t)}{G^*(t)}\,(1 - r^*(t))$$

The right-hand side of this equation follows from

$$1 - r^*(t) = 1 - \frac{\Pr(T^* = t)}{\Pr(T^* \geq t)} = \frac{\Pr(T^* > t)}{\Pr(T^* \geq t)} = \frac{G^*(t)}{G^*(t-1)} \tag{2}$$

This equation also implies an expression for the survivor function in terms of transition rates:

$$G^*(t) = \prod_{l=1}^{t} (1 - r^*(l))$$

We can now extend these concepts to situations with several transitions. The terminology is analogous to the continuous time case. There are origin states $j \in \mathcal{O}$ with destination states $k \in \mathcal{D}_j$; and with each origin state is associated a two-dimensional discrete random variable (T_j^*, D_j). The variable T_j^* describes the duration in the origin state j until an event occurs; and the variable D_j gives the destination state if an event occurs.

A description of this two-dimensional variable is given by the two-dimensional probability function

$$f_{jk}^*(t) = \Pr(T_j^* = t, D_j = k)$$

expressing the probability that a transition from origin state j to destination state k takes place at the discrete point in time t. Next, as in the continuous time case, transition-specific rates can be defined by

$$r_{jk}^*(t) = \Pr(T_j^* = t, D_j = k \mid T_j^* \geq t)$$

This is the conditional probability that a transition from origin state j to destination state k occurs at t, given the origin state j and that no event occurred until this point in time. It follows that the transition-specific rates add together to the overall exit rate

$$r_j^*(t) = \sum_{k \in \mathcal{D}_j} r_{jk}^*(t) \tag{3}$$

The associated survivor function now depends on the given origin state j and is defined by

$$G_j^*(t) = \Pr(T_j^* > t) \tag{4}$$

This is the probability that no event occurs up to, and including, the discrete point in time t. The transition rate from the given origin state j to destination state k may finally be written as

$$r^*_{jk}(t) = \frac{f^*_{jk}(t)}{G_j(t-1)}$$

Having introduced the basic concepts, the idea of transition rate models in discrete time can be described analogously to the continuous time case by

$$r^*_{jk}(t) = g_{jk}(t, x_{jk}, \beta_{jk}, \epsilon_{jk})$$

The basic idea is again to make the transition rate from origin state j to destination state k dependent on a vector of possibly transition-specific covariates x_{jk} with associated coefficients β_{jk}, and possibly on some not observed stochastic influences comprised in the variables ϵ_{jk}. The dependence is formulated by assuming a function g_{jk}; each different specification of this function gives a different discrete time rate model.

Maximum Likelihood Estimation. Before dealing with some possible model specifications we shortly describe maximum likelihood estimation in the discrete time framework. The approach is basically the same as in the continuous time case, but there are some small differences. The most important one is that the likelihood in the case of several transitions does not factorize into transition-specific terms.

1. In the case of a single transition, the likelihood may be written as

$$\mathcal{L} = \prod_{i \in \mathcal{E}} f^*(t_i) \prod_{i \in \mathcal{Z}} G^*(t_i)$$

where $\mathcal{E}$ denotes the set of episodes ending with an event, and $\mathcal{Z}$ denotes the set of censored episodes. The contribution to the likelihood of an episode that has an event at t_i is simply the probability that an event takes place at this point in time. The contribution of an episode that is censored at t_i is the value of the survivor function at t_i, the probability that no event has occurred until, and including, t_i.

2. In the case of a given origin state j, but possibly more than one destination state, the likelihood is

$$\mathcal{L}_j = \prod_{k \in \mathcal{D}_j} \prod_{i \in \mathcal{E}_{jk}} f^*_{jk}(t_i) \prod_{i \in \mathcal{Z}_j} G^*_j(t_i) \tag{5}$$

where $\mathcal{E}_{jk}$ is the set of episodes with origin state j and destination state k (implying that these episodes are not censored), and $\mathcal{Z}_j$ is the set of censored episodes with origin state j. The contribution of an uncensored episode to the likelihood is now the value of the tow-dimensional probability function $f^*_{jk}(t_i)$; the contribution of a censored episode is again the overall survivor function as defined in (4).

3. If there are several origin states, the total likelihood is simply the product of the likelihoods for each given origin state as defined in (5):

$$\mathcal{L} = \prod_{j\in\mathcal{O}} \prod_{k\in\mathcal{D}_j} \prod_{i\in\mathcal{E}_{jk}} f^*_{jk}(t_i) \prod_{i\in\mathcal{Z}_j} G^*_j(t_i) \tag{6}$$

This may be rewritten to reach a somewhat more convenient expression. By using (2) and (4) we get

$$f^*_{jk}(t) = \frac{r^*_{jk}(t)}{1 - r^*_j(t)}\, G^*_j(t)$$

and

$$G^*_j(t) = \prod_{l=1}^{t} (1 - r^*_j(l))$$

By substituting these expressions in (6) and taking the logarithm, the overall log-likelihood becomes

$$\ell = \sum_{j\in\mathcal{O}} \sum_{k\in\mathcal{D}_j} \sum_{i\in\mathcal{E}_{jk}} \log\left\{\frac{r^*_{jk}(t_i)}{1 - r^*_j(t_i)}\right\} + \sum_{i\in\mathcal{N}_j} \sum_{l=1}^{t_i} \log\left\{1 - r^*_j(l)\right\} \tag{7}$$

where $\mathcal{N}_j$ denotes the set of all episodes with origin state j. This formulation shows that we only need expressions for the transition rates to derive maximum likelihood estimates.

Episode Splitting. To include covariates, transition rates must be conditioned on their values. If covariates do not change during episodes the log-likelihood (7) can be used without modification. If covariates change their values at some discrete points in time the method of episode splitting can be used.

In order to apply this method, TDA uses conditional survivor functions, defined by

$$G^*_j(t \mid s) = \Pr\left(T^*_j > t \mid T^*_j > s\right) = \frac{G^*_j(t)}{G^*_j(s)}$$

with the convention that $G_j^*(0) = 1$. Substituting these conditional survivor functions in (7), the log likelihood becomes

$$\ell = \sum_{j \in \mathcal{O}} \sum_{k \in \mathcal{D}_j} \sum_{i \in \mathcal{E}_{jk}} \log \left\{ \frac{r_{jk}^*(t_i)}{1 - r_j^*(t_i)} \right\} + \sum_{i \in \mathcal{N}_j} \sum_{l=s_i+1}^{t_i} \log \left\{ 1 - r_j^*(l) \right\}$$

This modified log likelihood is used by TDA and provides for the possibility to use split episodes and to condition on given starting times not necessarily equal to zero.

6.17.6.2 Logistic Regression Models

This section describes the logistic regression model. It was discussed by several authors in somewhat different versions. Here we follow the exposition given by Allison [1982]. In the single transition case the model can be written as

$$r^*(t) = \frac{\exp(A\alpha + \beta_1 t + \beta_2 t^2 + \ldots + \beta_n t^n)}{1 + \exp(A\alpha + \beta_1 t + \beta_2 t^2 + \ldots + \beta_n t^n)} \quad t = 1, \ldots, q$$

A is a (row) vector of covariates with the first component equal to one, and α is a vector of associated coefficients. The time dependence is assumed to be a polynomial of order n in t. Therefore, the model is quite flexible, and one can test how many terms of the polynomial are needed for a sufficient fit.

Implementation. The logistic regression model has model number 20; the command to request model estimation is

```
rate (parameter) = 20;
```

For a description of the syntax and options see 6.17.1.4. Implementation of the model provides for possibly more than one transition and can be described by

$$r^*_{jk}(t) = \frac{a_{jk}\, b_{jk}(t)}{1 + a_{jk}\, b_{jk}(t)} \tag{1}$$

$$a_{jk} = \exp\left\{A^{(jk)}\alpha^{(jk)}\right\}$$

$$b_{jk}(t) = \exp\left\{\beta_1^{(jk)} t + \beta_2^{(jk)} t^2 + \ldots + \beta_n^{(jk)} t^n\right\}$$

There are two model terms. The product of a (row) vector $A^{(jk)}$ of possibly transition-specific covariates (with first component equal to one) and of associated coefficients $\alpha^{(jk)}$ is linked via an exponential link function to the first, the A-term, of the model. In addition, a transition-specific polynomial in t, with coefficient vector $\beta^{(jk)}$, is linked to the second, the B-term, of the model. Linking is again via an exponential link function.

The degree of the polynomial, n, can be specified with the `deg` parameter in the `rate` command; default is `deg=0`.

Maximum Likelihood Estimation. Model estimation follows the outline given in 6.17.6.1. Since the likelihood does not factor into transition-specific terms, mixed derivatives across different destination states must be included. First, following (3), the overall exit rate from origin state j is

$$r_j^*(t) = \sum_{k \in \mathcal{D}_j} r_{jk}^*(t)$$

where $r_{jk}^*(t)$ is given by (1). The log-likelihood for a given origin state j is

$$\ell_j = \sum_{k \in \mathcal{D}_j} \sum_{i \in \mathcal{E}_{jk}} \log \left\{ \frac{r_{jk}^*(t_i)}{1 - r_j^*(t_i)} \right\} + \sum_{i \in \mathcal{N}_j} \sum_{l=s_i+1}^{t_i} \log \left\{ 1 - r_j^*(l) \right\} \quad (2)$$

A small simplification is possible if we define a variable

$$t_i^* = \begin{cases} t_i - 1 & \text{if } i \in \mathcal{E}_j \\ t_i & \text{if } i \in \mathcal{Z}_j \end{cases}$$

where $\mathcal{E}_j$ and $\mathcal{Z}_j$ are, respectively, the set of uncensored and of censored episodes with origin state j. (2) may then be rewritten as

$$\ell_j = \sum_{k \in \mathcal{D}_j} \sum_{i \in \mathcal{E}_{jk}} \log \left\{ r_{jk}^*(t_i) \right\} + \sum_{i \in \mathcal{N}_j} \sum_{l=s_i+1}^{t_i^*} \log \left\{ 1 - r_j^*(l) \right\} \quad (3)$$

This log-likelihood is used for the formulation of derivatives. We denote the left-hand part of the right side, that is the additional part for uncensored episodes, by ℓ_j^L, the right-hand part, for all episodes, by ℓ_j^R. To write down the derivatives we will use the following abbreviations.

$$U_{jk}^L(t) = 1 - r_{jk}^*(t)$$

$$V_{jk}^L(t) = -r_{jk}^*(t) \left(1 - r_{jk}^*(t) \right)$$

$$U_{jk}^R(t) = -\frac{r_{jk}^*(t)(1 - r_{jk}^*(t))}{1 - r_j^*(t)}$$

$$V_{jk}^{R}(t) = \frac{r_{jk}^{*}(t)(1-r_{jk}^{*}(t))}{1-r_{j}^{*}(t)} \left[2\, r_{jk}^{*}(t) - 1 - \frac{r_{jk}^{*}(t)(1-r_{jk}^{*}(t))}{1-r_{j}^{*}(t)} \right]$$

$$W_{jkk'}^{R}(t) = \frac{-r_{jk}^{*}(t)(1-r_{jk}^{*}(t))\; r_{jk'}^{*}(t)(1-r_{jk'}^{*}(t))}{(1-r_{j}^{*}(t))^{2}}$$

The derivatives of the left part on the right-hand side of (3), ℓ_j^L, with respect to $\alpha^{(jk)}$ and $\beta^{(jk)}$ may then be written as

$$\frac{\partial \ell_j^L}{\partial \alpha_{j_a}^{(jk)}} = \sum_{k\in\mathcal{D}_j} \sum_{i\in\mathcal{E}_{jk}} U_{jk}^{L}(t_i)\, A_{j_a}^{(jk)}$$

$$\frac{\partial \ell_j^L}{\partial \beta_{j_b}^{(jk)}} = \sum_{k\in\mathcal{D}_j} \sum_{i\in\mathcal{E}_{jk}} U_{jk}^{L}(t_i)\, t_i^{j_b}$$

$$\frac{\partial^2 \ell_j^L}{\partial \alpha_{j_a}^{(jk)} \alpha_{k_a}^{(jk)}} = \sum_{k\in\mathcal{D}_j} \sum_{i\in\mathcal{E}_{jk}} V_{jk}^{L}(t_i)\, A_{j_a}^{(jk)} A_{k_a}^{(jk)}$$

$$\frac{\partial^2 \ell_j^L}{\partial \beta_{j_b}^{(jk)} \beta_{k_b}^{(jk)}} = \sum_{k\in\mathcal{D}_j} \sum_{i\in\mathcal{E}_{jk}} V_{jk}^{L}(t_i)\, t_i^{j_b+k_b}$$

$$\frac{\partial^2 \ell_j^L}{\partial \alpha_{j_a}^{(jk)} \beta_{k_b}^{(jk)}} = \sum_{k\in\mathcal{D}_j} \sum_{i\in\mathcal{E}_{jk}} V_{jk}^{L}(t_i)\, A_{j_a}^{(jk)} t_i^{k_b}$$

The derivatives of the right-hand side of (3), ℓ_j^R, with respect to $\alpha^{(jk)}$ and $\beta^{(jk)}$, are

$$\frac{\partial \ell_j^R}{\partial \alpha_{j_a}^{(jk)}} = \sum_{i\in\mathcal{N}_j} \sum_{l=s_i+1}^{t_i^*} U_{jk}^{R}(l)\, A_{j_a}^{(jk)}$$

$$\frac{\partial \ell_j^R}{\partial \beta_{j_b}^{(jk)}} = \sum_{i\in\mathcal{N}_j} \sum_{l=s_i+1}^{t_i^*} U_{jk}^{R}(l)\, l^{j_b}$$

$$\frac{\partial^2 \ell_j^R}{\partial \alpha_{j_a}^{(jk)} \alpha_{k_a}^{(jk)}} = \sum_{i\in\mathcal{N}_j} \sum_{l=s_i+1}^{t_i^*} V_{jk}^{R}(l)\, A_{j_a}^{(jk)} A_{k_a}^{(jk)}$$

$$\frac{\partial^2 \ell_j^R}{\partial \beta_{j_b}^{(jk)} \beta_{k_b}^{(jk)}} = \sum_{i \in \mathcal{N}_j} \sum_{l=s_i+1}^{t_i^*} V_{jk}^R(l)\, l^{j_b+k_b}$$

$$\frac{\partial^2 \ell_j^R}{\partial \alpha_{j_a}^{(jk)} \beta_{k_b}^{(jk)}} = \sum_{i \in \mathcal{N}_j} \sum_{l=s_i+1}^{t_i^*} V_{jk}^R(l)\, A_{j_a}^{(jk)} l^{k_b}$$

Finally, the mixed derivatives, $k \neq k'$, are

$$\frac{\partial^2 \ell_j^R}{\partial \alpha_{j_a}^{(jk)} \alpha_{k_a}^{(jk')}} = \sum_{i \in \mathcal{N}_j} \sum_{l=s_i+1}^{t_i^*} W_{jkk'}^R(l)\, A_{j_a}^{(jk)} A_{k_a}^{(jk')}$$

$$\frac{\partial^2 \ell_j^R}{\partial \beta_{j_b}^{(jk)} \beta_{k_b}^{(jk')}} = \sum_{i \in \mathcal{N}_j} \sum_{l=s_i+1}^{t_i^*} W_{jkk'}^R(l)\, l^{j_b+k_b}$$

$$\frac{\partial^2 \ell_j^R}{\partial \alpha_{j_a}^{(jk)} \beta_{k_b}^{(jk')}} = \sum_{i \in \mathcal{N}_j} \sum_{l=s_i+1}^{t_i^*} W_{jkk'}^R(l)\, A_{j_a}^{(jk)} l^{k_b}$$

Initial Estimates. Without duration-dependence (`deg=0`), the logistic regression model is similar to the exponential model and TDA uses identical default starting values. If the polynomials to capture duration-dependence have higher degrees these default starting values can easily lead to numerical problems and convergence difficulties. A sensible strategy is to estimate a series of models with increasing degrees of the polynomial and to use the parameter estimates of previously estimated models to define starting values.

Example 1 To illustrate the logistic regression model, we use our main episode data set (`rrdat.1`) described in **3.3.3**. Box 1 shows the command file, `rtd.1`, to estimate a logistic regression model without duration-dependence. Standard output is shown in Box 2.

To estimate models with duration dependence, one can add the parameter

```
deg = degree_of_polynomial,
```

to the rate command. As we have mentioned, higher degree polynomials can lead to numerical difficulties due to very different magnitudes of the model parameters. Also, if there are many time points, the estimation process can be very slow.

Box 1 Command file `rtd1.cf` for logistic regression

```
nvar(
    dfile = rrdat.1,   # data file

    ID    [3.0] = c1,  # identification number
    SN    [2.0] = c2,  # spell number
    TS    [3.0] = c3,  # starting time
    TF    [3.0] = c4,  # ending time
    SEX   [2.0] = c5,  # sex (1 men, 2 women)
    TI    [3.0] = c6,  # interview date
    TB    [3.0] = c7,  # birth date
    TE    [3.0] = c8,  # entry into labor market
    TMAR  [3.0] = c9,  # marriage date (0 if no marriage)
    PRES  [3.0] = c10, # prestige of current job
    PRESN [3.0] = c11, # prestige of next job
    EDU   [2.0] = c12, # highest educational attainment

    COHO1 = ge(TB,348) & le(TB,384), # birth cohort 1
    COHO2 = ge(TB,468) & le(TB,504), # birth cohort 2
    COHO3 = ge(TB,588) & le(TB,624), # birth cohort 3
    W     = SEX[2],                  # women = 1

    DES [1.0] = if eq(TF,TI) then 0 else 1,  # destination state
    DUR [3.0] = TF - TS + 1,
);
edef(
    ts = 0,     # starting time
    tf = DUR,   # ending time
    org = 0,    # origin state
    des = DES,  # destination state
);
rate (
    xa(0,1) = COHO2,COHO3,W,

) = 20;
```

Example 2 The TDA example archive contains another command file, `rtd1m.cf`, that can be used to estimate logistic regression models for episodes with three alternative destination states. It is basically identical with command file `rt1m.cf` described in 6.17.2.1.

Additional Remarks. TDA cannot calculate generalized residuals for the logistic regression model and the `pres` option will be ignored. Also the `rrisk` and `prate` parameters cannot be used with this model.

Box 2 Standard output from command file `rtd1.cf`

```
Model: Discrete time logistic regression, degree 0.

Maximum likelihood estimation.
Algorithm 5: Newton (I)

Number of model parameters: 4
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Log-likelihood of exponential null model: -2514.02
Changed scaling factor for log-likelihood: -0.001
Using default starting values.

Convergence reached in 5 iterations.
Number of function evaluations: 6 (6,6)

Maximum of log likelihood: -2472.39
Norm of final gradient vector: 1.04739e-07
Last absolute change of function value: 2.93992e-11
Last relative change in parameters: 3.51238e-05

Idx SN Org Des MT Variable    Coeff    Error    C/Error  Signif
----------------------------------------------------------------
  1  1   0   1  A Constant  -5.0056   0.0847   -59.1117  1.0000
  2  1   0   1  A COHO2      0.5402   0.1127     4.7944  1.0000
  3  1   0   1  A COHO3      0.6822   0.1160     5.8808  1.0000
  4  1   0   1  A W          0.5133   0.0949     5.4097  1.0000

Log likelihood (starting values): -2511.4674
Log likelihood (final estimates): -2472.3890
```

6.17.6.3 Complementary Log-Log Models

Following again the exposition by Allison [1982], this section describes the complementary log-log model. In the single transition case, the transition rate is

$$r^*(t) = 1 - \exp\left\{-\exp(A\alpha + \beta_1 t + \beta_2 t^2 + \ldots + \beta_n t^n)\right\}$$

Implementation. The command to request model estimation is

```
rate (parameter) = 21;
```

For a description of the syntax and options see 6.17.1.4. Implementation of the model provides for possibly more than one transition and can be described by

$$r^*_{jk}(t) = 1 - \exp\left\{-a_{jk}\, b_{jk}(t)\right\}$$

$$a_{jk} = \exp\left\{A^{(jk)}\alpha^{(jk)}\right\}$$

$$b_{jk}(t) = \exp\left\{\beta_1^{(jk)} t + \beta_2^{(jk)} t^2 + \ldots + \beta_n^{(jk)} t^n\right\}$$

Again, there are two model terms. The product of a (row) vector $A^{(jk)}$ of possibly transition-specific covariates (with the first component equal to one), and of associated coefficients $\alpha^{(jk)}$, is linked via an exponential link function to the first, the A-term, of the model. In addition, a transition-specific polynomial in t, with coefficient vector $\beta^{(jk)}$, is linked to the second, the B-term, of the model. The degree of the polynomial can be specified with the `deg` parameter in the `rate` command; default is `deg=0`.

Maximum Likelihood Estimation. Model estimation follows the outline given in 6.17.6.1 and is similar to the estimation procedure for the logistic regression model. Using the notation introduced in 6.17.6.2, the log-likelihood for the complementary log-log model can be written as

$$\ell_j = \sum_{k\in\mathcal{D}_j}\sum_{i\in\mathcal{E}_{jk}} \log\left\{r^*_{jk}(t_i)\right\} + \sum_{i\in\mathcal{N}_j}\sum_{l=s_i+1}^{t_i^*} \log\left\{1 - r^*_j(l)\right\}$$

Also the expressions for the derivatives are the same as for the logistic regression model, one only has to change the abbreviations defined in 6.17.6.2 into

$$U^L_{jk}(t) = a_{jk}\, b_{jk}(t) \left[\frac{1}{r^*_{jk}(t)} - 1 \right]$$

$$V^L_{jk}(t) = a_{jk}\, b_{jk}(t) \left[\frac{1}{r^*_{jk}(t)} - 1 \right] \left[1 - \frac{a_{jk}\, b_{jk}(t)}{r^*_{jk}(t)} \right]$$

$$U^R_{jk}(t) = a_{jk}\, b_{jk}(t)\, \frac{r^*_{jk}(t) - 1}{1 - r^*_j(t)}$$

$$V^R_{jk}(t) = a_{jk}\, b_{jk}(t)\, \frac{1 - r^*_{jk}(t)}{1 - r^*_j(t)} \left[a_{jk}\, b_{jk}(t)\, \frac{r^*_{jk}(t) - r^*_j(t)}{1 - r^*_j(t)} - 1 \right]$$

$$W^R_{jkk'}(t) = \frac{-a_{jk}\, b_{jk}(t)(1 - r^*_{jk}(t))\; a_{jk'} b_{jk'}(t)(1 - r^*_{jk'}(t))}{(1 - r^*_j(t))^2}$$

Using these abbreviations, the first and second derivatives of the log likelihood have the same formulation as described in 6.17.6.2.

Initial Estimates. By default, TDA uses the default starting values for an exponential null model. Similar to the logistic regression model, there can occur numerical difficulties if duration dependence is specified by a higher degree polynomial.

Example 1 To illustrate estimation of a complementary log-log model we use command file `rtd2.cf` (not shown). This command file identical to `rtd1.cf` (see 6.17.6.2), only the model number was changed into 21. Part of the standard output is shown in Box 1.

The TDA example archive contains another command file, `rtd2m.cf`, which can be used to estimate a complementary log-log model for episodes with alternative destination states.

Additional Remarks. As with the logistic regression mode, TDA cannot calculate generalized residuals for the complementary log-log model and the `pres` option will be ignored. Also the `rrisk` and `prate` parameters are ignored.

Box 1 Part of standard output from command file `rtd2.cf`

```
Model: Discrete time complementary log-log, degree 0.

Maximum likelihood estimation.
Algorithm 5: Newton (I)

Number of model parameters: 4
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Log-likelihood of exponential null model: -2514.02
Changed scaling factor for log-likelihood: -0.001
Using default starting values.

Convergence reached in 5 iterations.
Number of function evaluations: 6 (6,6)

Maximum of log likelihood: -2472.42
Norm of final gradient vector: 1.31763e-07
Last absolute change of function value: 3.60281e-11
Last relative change in parameters: 3.83396e-05

Idx SN Org Des MT Variable    Coeff    Error    C/Error  Signif
----------------------------------------------------------------
  1  1   0   1  A Constant  -5.0081   0.0843   -59.4116  1.0000
  2  1   0   1  A COHO2      0.5371   0.1120     4.7959  1.0000
  3  1   0   1  A COHO3      0.6773   0.1152     5.8768  1.0000
  4  1   0   1  A W          0.5094   0.0942     5.4044  1.0000

Log likelihood (starting values): -2511.4459
Log likelihood (final estimates): -2472.4212
```

6.17.7 Semi-parametric Rate Models

This chapter describes a semi-parametric transition rate model proposed by Cox [1972].

6.17.7.1 Partial Likelihood Estimation

The transition rate models described in **6.17.3** are based on parametrical assumptions about the distribution of episode durations. This allows for straightforward maximum likelihood estimation. However, one not always has sufficient reasons to choose a specific model. The best strategy seems then to use a variety of different model specifications and to look for similar outcomes.

An alternative was proposed by Cox [1972] and subsequently discussed by many other authors. For the single transition case, and based on a continuous time variable $t \geq 0$, the model, generally called a Cox model, is

$$r(t) = h(t) \exp\left(A(t)\,\alpha\right) \tag{1}$$

The transition rate $r(t)$ depends on an unspecified baseline rate, $h(t)$, and on a vector of covariates $A(t)$ with coefficients α.[1] The covariates may depend on the process time, t.

Implementation. Implementation of the Cox model in TDA is based on the following model formulation.

$$r_{jk}(t) = h_{jk}(t)\, \exp\left\{A^{(jk)}(t)\,\alpha^{(jk)}\right\} \tag{2}$$

$r_{jk}(t)$ is the transition rate at time t for the transition from origin state j to destination state k. $h_{jk}(t)$ is the unspecified baseline rate for the same transition.[2] $A^{(jk)}(t)$ is a (row) vector of covariates, specified for the transition (j,k), and $\alpha^{(jk)}$ is a vector of associated coefficients. The covariates may have time-dependent values, examples will be given in **6.17.7.2** and **6.17.7.3**.

Furthermore, it is possible to explain each transition by a specific set of covariates. One should note, however, that model estimation requires that *all* covariates are defined for all members of a sample of episodes.

[1] We use the convention to write covariates as row vectors, the accompanying coefficients as column vectors.

[2] The formulation implies that the baseline rates are allowed to vary between different transitions. A special type of Cox models arises if the baseline rates are constrained to be identical for all transitions; cf. Kalbfleisch and Prentice [1980], p. 170.

This is necessary to calculate the risks for all transitions, regardless of the transition that actually happens at the end of an episode. One should also note that it is necessary to include at least one covariate for each transition because Cox models do not contain any intercept terms; all constant effects are included in the baseline rates.

Model estimation is based on the method of partial likelihood (Cox [1972], [1975]). To explain this method, we will first consider the case of a single transition (j, k) from origin state j to destination state k. Indices referring to transitions are dropped to simplify notation. With the assumption of no ties, i.e. all ending times in the sample are different, the partial likelihood, L^p, may be written as

$$L^p = \prod_{i \in \mathcal{E}} \frac{\exp(A_i(t_i)\,\alpha)}{\sum_{l \in \mathcal{R}(t_i)} \exp(A_l(t_i)\,\alpha)} \tag{3}$$

$\mathcal{E}$ denotes the set of all not censored episodes belonging to the (single) transition (j, k), i.e. having an event at their ending time. $\mathcal{R}(t_i)$ is the risk set at the ending time, say t_i, of the ith episode contained in $\mathcal{E}$. The definition of the risk set is exactly the same as was given for the product-limit estimator: the set of all episodes, exiting with an event or censored, with starting time less than t_i and ending time equal to or greater than t_i. $A_i(t_i)$ is the (row) vector of covariates for the ith episode (evaluated at t_i), and α is the vector of associated coefficients to be estimated.

Given this notation, the calculation of the partial likelihood is easily understood: One looks at the ending times of episodes ending with an event, then, for each of these points in time, the risk set is created, and finally the expression on the right-hand side of (3) is evaluated. This expression may be interpreted as the probability that it is just the ith individual to have an event at this point in time, given the risk set containing all individuals who could have an event. Note that in the calculation of these probabilities time-dependent covariates can be accounted for in a simple way: at each successive point in time the actual covariate values can be used.

As was shown by Cox and other authors, the partial likelihood defined in (3) may be treated as if it was a standard likelihood. Estimates of model coefficients reached by maximizing the partial likelihood have properties similar to those of standard maximum likelihood estimates.

The only difficulty arises if there are tied ending times. It is complicated, then, to calculate the partial likelihood exactly. Therefore several simplifying approximations have been proposed. As is done, for instance,

by BMDP and SAS, the TDA algorithm for the partial likelihood calculation uses an approximation proposed by Breslow [1974].[3]

Assume there are d_i episodes, with indices contained in a set D_i, all having the same ending time, t_i. The partial likelihood factor for this point in time is approximated, then, by

$$\frac{\exp(S_i(t_i)\,\alpha)}{\left[\sum_{l\in\mathcal{R}(t_i)}\exp(A_l(t_i)\,\alpha)\right]^{d_i}} = \prod_{j\in D_i}\frac{\exp(A_j(t_i)\,\alpha)}{\sum_{l\in\mathcal{R}(t_i)}\exp(A_l(t_i)\,\alpha)}$$

with $S_i(t_i)$ defined as the sum of the covariate vectors for the d_i episodes with events at t_i (evaluated at t_i). As is easily seen, using this approximation is already accounted for in the formulation of the partial likelihood in (3), because in this formulation the product goes over all episodes, tied or not, contained in $\mathcal{E}$.

As is generally assumed, this approximation is sufficient if there are only "relatively few" ties in a sample of episode data. If there are "too many" ties it seems preferable to use a discrete time model. Box 1 shows how the partial likelihood calculation is done in TDA. The algorithm includes the possibility of stratification variables and time-dependent covariates explained in later sections.

Alternative Destination States. To extend the Cox model for sets of episode data with several transitions, we follow the general remarks in **3.3.1**. First it should be restated that, in the context of duration analysis, all estimations are performed conditional on given origin states. Therefore, with two or more origin states, the (partial) likelihood may be built separately for all origin states; each single term may be maximized separately, or the product of all origin-specific terms is taken and maximized simultaneously.[4]

Also the case of two or more transitions, from a given origin state, is simple.[5] One only needs an appropriate definition of the risk set. The risk set is now conditional on a given origin state, so we shall write $\mathcal{R}_j(t)$; but one has to include all episodes, with given origin state j, that are at

[3] We follow the formulation given by Lawless [1982], p. 346. An introduction to Breslow's approximation may also be found in Namboodiri and Suchindran [1987], p. 213.

[4] The algorithm in TDA maximizes the partial likelihood simultaneously for all transitions to provide for the possibility of constraints on model parameters across transitions. In this case the transition-specific terms of the likelihood are no longer independent, and the maximization must be done simultaneously.

[5] Cf. Kalbfleisch and Prentice [1980], p. 169.

Box 1 Partial Likelihood Algorithm in TDA

```
For all transitions: TRAN = 1,...,NT {
    ORG = origin state of transition TRAN
    DES = destination state of transition TRAN
    For all groups: G = 1,...,NG {
        TIME = 0
        For all episodes: EP = 1,...,NE (according to their ending times) {
            ORG1 = origin state of episode EP
            DES1 = destination state of episode EP
            TF = ending time of episode EP
            If (ORG1 = ORG and DES1 = DES and EP belongs to group G) {
                If (no time-dependent covariates) {
                    Add to the risk set all episodes which belong to
                    group G, have origin state org, and starting time
                    less than TF.
                }
                If (TF greater than TIME) {
                    If (no time-dependent covariates) {
                        Eliminate from the risk set all episodes which
                        belong to group G, have origin state ORG,
                        and have ending time less than TF.
                    }
                    If (time-dependent covariates) {
                        Create a new risk set: all episodes that belong
                        to group G, have origin state ORG, starting
                        time less than TF and ending time greater or
                        equal to TF. While creating this new risk
                        set, use the values of time-dependent covariates
                        at TF.
                    }
                    TIME = TF
                }
                Update partial likelihood (and derivatives) with
                episode EP.
            }
        }
    }
}
```

Note: If no groups (strata) are defined, this algorithm assumes that there is exactly one group containing all episodes.

risk for a transition to state k at time t, regardless of which destination state they actually will reach at some later point in time. Therefore, the risk set $\mathcal{R}_j(t)$ is defined as the set of *all* episodes with origin state j provided that their starting time is less than t and their ending time is equal to or greater than t.

The partial likelihood in the case of possibly more than one origin and/or destination state may then be written as

$$L^p = \prod_{j\in\mathcal{O}} \prod_{k\in\mathcal{D}_j} \prod_{i\in\mathcal{E}_{jk}} \frac{\exp\left(A^{(jk)}(t_i)\,\alpha^{(jk)}\right)}{\sum_{l\in\mathcal{R}_j(t_i)} \exp\left(A^{(jk)}(t_l)\,\alpha^{(jk)}\right)} \tag{4}$$

Box 1 shows the implementation of this partial likelihood calculation in TDA. If there is only one transition this reduces to the partial likelihood given in (3). Also the approximation, if there are tied ending times, is the same as in the single transition case.

Maximizing the Partial Likelihood. Maximum partial likelihood estimates are found by maximizing (4). Dropping indices referring to transitions and time-dependence of covariates, the logarithm of (4) may be written as

$$\ell^p = \sum_{i\in\mathcal{E}} A_i\,\alpha - \log\left\{\sum_{l\in\mathcal{R}_i} \exp(A_l\,\alpha)\right\} \tag{5}$$

Denoting by $A_{i,j}$ the jth element of the covariate vector for the episode $i \in \mathcal{E}$, the first and second derivatives are

$$\frac{\partial \ell^p}{\partial \alpha_{j_a}} = \sum_{i\in\mathcal{E}} A_{i,j_a} - \frac{\sum_{l\in\mathcal{R}_i} \exp(A_l\,\alpha)\,A_{l,j_a}}{\sum_{l\in\mathcal{R}_i} \exp(A_l\,\alpha)}$$

$$\frac{\partial^2 \ell^p}{\partial \alpha_{j_a}\partial \alpha_{k_a}} = \sum_{i\in\mathcal{E}} \frac{\sum_{l\in\mathcal{R}_i} \exp(A_l\,\alpha)\,A_{l,j_a}}{\sum_{l\in\mathcal{R}_i} \exp(A_l\,\alpha)} \frac{\sum_{l\in\mathcal{R}_i} \exp(A_l\,\alpha)\,A_{l,k_a}}{\sum_{l\in\mathcal{R}_i} \exp(A_l\,\alpha)} - \frac{\sum_{l\in\mathcal{R}_i} \exp(A_l\,\alpha)\,A_{l,j_a}A_{l,k_a}}{\sum_{l\in\mathcal{R}_i} \exp(A_l\,\alpha)}$$

Example 1 To illustrate partial likelihood estimation with TDA, the first example will be based on our main example data set, `rrdat.1`, described in **3.3.3**. For this example we only consider a single transition and select two covariates: birth cohort and sex. Estimation is done with command file `pl1.cf`, basically identical with the command files used in previous chapters. Estimation results are shown in Box 3.

Box 2 Command file `pl1.cf`

```
nvar(
  dfile = rrdat.1,   # data file

  ID    [3.0] = c1,  # identification number
  SN    [2.0] = c2,  # spell number
  TS    [3.0] = c3,  # starting time
  TF    [3.0] = c4,  # ending time
  SEX   [2.0] = c5,  # sex (1 men, 2 women)
  TI    [3.0] = c6,  # interview date
  TB    [3.0] = c7,  # birth date
  TE    [3.0] = c8,  # entry into labor market
  TMAR  [3.0] = c9,  # marriage date (0 if no marriage)
  PRES  [3.0] = c10, # prestige of current job
  PRESN [3.0] = c11, # prestige of next job
  EDU   [2.0] = c12, # highest educational attainment

  COHO1 = ge(TB,348) & le(TB,384), # birth cohort 1
  COHO2 = ge(TB,468) & le(TB,504), # birth cohort 2
  COHO3 = ge(TB,588) & le(TB,624), # birth cohort 3
  W     = SEX[2],                  # women = 1

  DES [1.0] = if eq(TF,TI) then 0 else 1,  # destination state
  DUR [3.0] = TF - TS + 1,
);

edef(
  ts = 0,     # starting time
  tf = DUR,   # ending time
  org = 0,    # origin state
  des = DES,  # destination state
);

rate (
  xa(0,1) = COHO2,COHO3,W,

) = 1;      model number 1 for Cox model
```

Example 2 For a second example, we distinguish three destination states. The command file is `pl1m.cf`, estimation results are shown in Box 4. Actually, one would get the same result if estimating separate models for each of the three destination states taking into account all other episodes as censored.

Example 3 Lawless ([1982], p. 367) has given an example for the Cox model with some biometrical data. This example can be replicated with

Box 3 Output from command file `pl1.cf`

```
Model: Cox (partial likelihood)

Maximum Likelihood Estimation.
Algorithm 5: Newton (I)

Number of model parameters: 3
Type of covariance matrix: 2
Maximum number of iterations: 20
Convergence criterion: 1
Tolerance for norm of final gradient: 1e-06
Mue of Armijo condition: 0.2
Minimum of step size value: 1e-10
Scaling factor: -1

Log-likelihood of exponential null model: -2514.02
Scaling factor for log-likelihood: -0.001
Using default starting values.
Sorting episodes according to ending times.

Convergence reached in 4 iterations.
Number of function evaluations: 5 (5,5)

Maximum of log likelihood: -2565.43
Norm of final gradient vector: 8.94115e-09
Last absolute change of function value: 1.45822e-11
Last relative change in parameters: 5.14129e-05

Idx SN Org Des MT Variable      Coeff      Error    C/Error  Signif
-------------------------------------------------------------------
  1  1   0   1  A COHO2        0.3797     0.1134     3.3501  0.9992
  2  1   0   1  A COHO3        0.4585     0.1177     3.8937  0.9999
  3  1   0   1  A W            0.4078     0.0949     4.2988  1.0000

Log likelihood (starting values): -2584.5701
Log likelihood (final estimates): -2565.4257
```

command file `pl2.cf`, contained in the TDA example archive.

Box 4 Part of output from command file `pl1m.cf`

```
Model: Cox (partial likelihood)

Convergence reached in 4 iterations.
Number of function evaluations: 5 (5,5)

Maximum of log likelihood: -2550.89
Norm of final gradient vector: 1.07487e-06
Last absolute change of function value: 1.9399e-09
Last relative change in parameters: 0.000489307

Idx SN Org Des MT Variable      Coeff      Error    C/Error  Signif
-------------------------------------------------------------------
  1  1   0   1  A COHO2         0.0893     0.2640     0.3381  0.2647
  2  1   0   1  A COHO3         0.1624     0.2696     0.6024  0.4531
  3  1   0   1  A W            -0.2411     0.2352    -1.0254  0.6948
-------------------------------------------------------------------
  4  1   0   2  A COHO2         0.5017     0.1641     3.0564  0.9978
  5  1   0   2  A COHO3         0.5487     0.1714     3.2010  0.9986
  6  1   0   2  A W             0.1879     0.1387     1.3547  0.8245
-------------------------------------------------------------------
  7  1   0   3  A COHO2         0.3726     0.1964     1.8966  0.9421
  8  1   0   3  A COHO3         0.4909     0.2039     2.4069  0.9839
  9  1   0   3  A W             1.0651     0.1678     6.3482  1.0000

Log likelihood (starting values): -2584.5701
Log likelihood (final estimates): -2550.8900
```

6.17.7.2 Time-dependent Covariates

There is an easy way to include time-dependent covariates into Cox models. The method is based on the fact that the partial likelihood calculation goes gradually through all points in time where at least one of the uncensored episodes has an event. It is possible, then, to re-evaluate the values of time-dependent covariates at these time points. Actually, this is already provided for in the partial likelihood formulas given in 6.17.7.1. As shown by the algorithm in Box 6.17.7.1-1 the risk set is re-calculated at every new point in time.

To define time-dependent covariates, TDA provides a special operator, `time`. Note that this operator is only valid for type 5 variables defined inside the `edef` command. All type 5 variables are re-evaluated during partial likelihood estimation substituting `time` by the current process time.

Example 1 To illustrate time-dependent covariates we continue with Example 6.17.7.1-1 and investigate the hypothesis that marriage has some influence on job duration. Box 1 shows command file `pl3.cf` using a time-dependent dummy variable `MARR`. Estimation results are shown in Box 2.

One should note that estimation results, when using time-dependent covariates, can depend critically on the point in time when the variable changes its value, in particular when the data are heavily tied. In general, one should expect different results if using the `gt` or the `ge` operator in defining the time-dependent variable.[1] For practical applications, one should also consider the introduction of lags, see the discussion of "effect shapes" in Blossfeld et al. [1996].

[1]Since process time in TDA is treated as a floating point number, there is also a problem of rounding errors. Different from previous versions of TDA, the evaluation of the logical operators `ge`, `gt`, etc. now uses a "safe" comparison that tries to minimize rounding errors.

Box 1 Command file `pl3.cf`

```
nvar(
  dfile = rrdat.1,   # data file

  ID    [3.0] = c1,  # identification number
  SN    [2.0] = c2,  # spell number
  TS    [3.0] = c3,  # starting time
  TF    [3.0] = c4,  # ending time
  SEX   [2.0] = c5,  # sex (1 men, 2 women)
  TI    [3.0] = c6,  # interview date
  TB    [3.0] = c7,  # birth date
  TE    [3.0] = c8,  # entry into labor market
  TMAR  [3.0] = c9,  # marriage date (0 if no marriage)
  PRES  [3.0] = c10, # prestige of current job
  PRESN [3.0] = c11, # prestige of next job
  EDU   [2.0] = c12, # highest educational attainment

  COHO1 = ge(TB,348) & le(TB,384), # birth cohort 1
  COHO2 = ge(TB,468) & le(TB,504), # birth cohort 2
  COHO3 = ge(TB,588) & le(TB,624), # birth cohort 3
  W     = SEX[2],                  # women = 1

  MDATE = if le(TMAR,0) then 10000 else TMAR - TS, # marriage date

  DES [1.0] = if eq(TF,TI) then 0 else 1,  # destination state
  DUR [3.0] = TF - TS + 1,
);

edef(
  ts = 0,     # starting time
  tf = DUR,   # ending time
  org = 0,    # origin state
  des = DES,  # destination state
  MARR  = gt(time,MDATE), # time-dependent dummy for married
);

rate (
  xa(0,1) = COHO2,COHO3,W,MARR,

) = 1;
```

Box 2 Part of standard output from command file `pl3.cf`

```
Model: Cox (partial likelihood)

Maximum likelihood estimation.
Algorithm 5: Newton (I)

Maximum of log likelihood: -2562.51
Norm of final gradient vector: 1.73888e-08
Last absolute change of function value: 2.16271e-11
Last relative change in parameters: 7.73306e-05

Idx SN Org Des MT Variable    Coeff      Error    C/Error  Signif
-------------------------------------------------------------------
  1  1   0   1  A COHO2       0.3920     0.1135     3.4548  0.9994
  2  1   0   1  A COHO3       0.4668     0.1179     3.9604  0.9999
  3  1   0   1  A W           0.3947     0.0950     4.1543  1.0000
  4  1   0   1  A MARR       -0.2408     0.1000    -2.4083  0.9840

Log likelihood (starting values): -2584.5701
Log likelihood (final estimates): -2562.5089
```

6.17.7.3 Episode Splitting

A disadvantage of the standard method to include time-dependent variables into Cox models is that the partial likelihood calculation becomes very time-consuming. The reason is that the risk set must be re-calculated, and the values of the covariates must be re-evaluated (including comparisons with the process time), for every point in time where at least one event occurs.

If time-dependent covariates change their values only at some discrete points in time, one can use, instead, the method of episode splitting. The original episodes are split at every time point where one of the time-dependent covariates changes its value. Each of the original episodes is substituted by a set of sub-episodes (splits) with appropriate values of the covariates. The last of these splits has the same exit status as the original episode, all other splits are regarded as right censored.

To use this method with partial likelihood maximization, it is only necessary to be precise in the definition of the risk set. At every time point t, the risk set should contain all episodes, or splits, which have a starting time less than t and ending time greater than, or equal to t, regardless of their exit status. Consequently, the risk set at every point in time contains only a single split—associated with appropriate values of the time-dependent covariates—for each of the original episodes. Clearly, it is necessary that the partial likelihood algorithm takes into account both the different ending *and* starting times of the episodes (splits). As shown in Box 1 in 6.17.7.1, this is a general feature of TDA's partial likelihood algorithm. (The method is inspired by the program RATE (Tuma [1980]) which, we presume, was the first using starting and ending times for updating the risk set.)

Unfortunately, one cannot use TDA's method of *internal* episode splitting with the partial likelihood algorithm. This is due to the fact that, to save storage space, episode splits are only virtually created. However, to be sorted according to starting and ending times, they must be present in the program's data matrix. Consequently, to use the method of episode splitting to estimate Cox models, or analogously for product-limit estimates, one has to proceed in two steps.

1. The first step is to create a new data file containing split episodes

Box 1 Command file `pl4.cf` for episode splitting

```
nvar(
    dfile = rrdat.1,   # data file

    ID    [3.0] = c1,  # identification number
    SN    [2.0] = c2,  # spell number
    TS    [3.0] = c3,  # starting time
    TF    [3.0] = c4,  # ending time
    SEX   [2.0] = c5,  # sex (1 men, 2 women)
    TI    [3.0] = c6,  # interview date
    TB    [3.0] = c7,  # birth date
    TE    [3.0] = c8,  # entry into labor market
    TMAR  [3.0] = c9,  # marriage date (0 if no marriage)
    PRES  [3.0] = c10, # prestige of current job
    PRESN [3.0] = c11, # prestige of next job
    EDU   [2.0] = c12, # highest educational attainment

    COHO1 = ge(TB,348) & le(TB,384), # birth cohort 1
    COHO2 = ge(TB,468) & le(TB,504), # birth cohort 2
    COHO3 = ge(TB,588) & le(TB,624), # birth cohort 3
    W     = SEX[2],                  # women = 1

    MDATE = if le(TMAR,0) then 10000 else TMAR - TS, # marriage date

    DES [1.0] = if eq(TF,TI) then 0 else 1,  # destination state
    DUR [3.0] = TF - TS + 1,
);
edef(
    ts = 0,          # starting time
    tf = DUR,        # ending time
    org = 0,         # origin state
    des = DES,       # destination state
    split = MDATE,   # split with MDATE
);
epdat(
    v = COHO1,COHO2,COHO3,W,MDATE,    # additional variables
    dtda = pl4.tda,                   # description file
) = pl4.dat;
```

instead of the original ones. This can be done with the `split` parameter in the `edef` command and then writing the new data file with the `epdat` command. Time-dependent covariates can be generated during this first step, but equally well based on the resulting output file containing the episode splits.

2. In a second step, the data file containing the episode splits can be used

Box 2 First records in data file `pl4.dat`

ID	SN	NSPL	SPL	ORG	DES	TS	TF	COHO1	COHO2	COHO3	W	MDATE
1	1	2	1	0	0	0.00	124.00	1	0	0	0	124
1	1	2	2	0	0	124.00	428.00	1	0	0	0	124
2	1	1	1	0	1	0.00	46.00	1	0	0	1	169
3	1	1	1	0	1	0.00	34.00	1	0	0	1	123
4	1	2	1	0	0	0.00	89.00	1	0	0	1	89
4	1	2	2	0	1	89.00	220.00	1	0	0	1	89
5	1	1	1	0	1	0.00	12.00	0	1	0	1	182
6	1	1	1	0	1	0.00	30.00	0	1	0	1	170
7	1	1	1	0	1	0.00	12.00	0	1	0	1	140
8	1	1	1	0	1	0.00	75.00	0	1	0	1	128
9	1	1	1	0	1	0.00	12.00	0	1	0	1	53
10	1	1	1	0	1	0.00	55.00	0	0	1	1	0
11	1	1	1	0	1	0.00	68.00	1	0	0	0	118
12	1	2	1	0	0	0.00	50.00	1	0	0	0	50
12	1	2	2	0	1	50.00	137.00	1	0	0	0	50

as if it was a set of "original" episodes. Based on the new input data, one can estimate Cox models and, of course, all of the parametric transition rate models discussed in **6.17.2** and **6.17.3**. It is also possible to request product-limit estimates based on a set of episode splits; one should get exactly the same results as if one had used the original episodes.

Example 1 To illustrate the method of episode splitting, we replicate Example 1 in **6.17.7.2**. The first step is to create a new data file containing episode splits, with splitting at the date of marriage. This is done with the command file `pl4.cf` shown in Box 1. There are three commands. The first command (`nvar`) reads the data file and creates the basic variables. The following `edef` command defines episode data and requests internal episode splitting with variable `MDATE`. Finally, the `epdat` command writes the new episode data into the output file `pl4.dat` and creates a data description file with the `dtda` parameter. The first records of the new data file, `pl4.dat`, are shown in Box 2. This new data file is then used by command file `pl5.cf` shown in Box 3.[1] The `nvar` command reads the data file, creates the basic variables and, in addition, the time-dependent dummy variable `MARR`. This variable gets the value 1 for each split where the marriage data is less than, or equal to the starting time of the split. The `edef` command specifies the episode data structure and creates the time-dependent dummy variable `MARR1` as

[1] This command file was created by using the description file `pl4.tda`.

Box 3 Command file `pl5.cf`

```
nvar(
  dfile = pl4.dat,
  noc = 761,
  ID   [6.0] = c1, # id number
  SN   [3.0] = c2, # spell number
  NSPL [3.0] = c3, # number of splits
  SPL  [3.0] = c4, # split number
  ORG  [3.0] = c5, # origin state
  DES  [3.0] = c6, # destination state
  TS   [6.2] = c7, # starting time
  TF   [6.2] = c8, # ending time
  COHO1 [0.0] = c9,
  COHO2 [0.0] = c10,
  COHO3 [0.0] = c11,
  W [0.0] = c12,
  MDATE [0.0] = c13,
  MARR = le(MDATE,TS),
);
edef(
    ts = TS,    # starting time
    tf = TF,    # ending time
    org = ORG,  # origin state
    des = DES,  # destination state
    MARR1 = le(MDATE,ts),
);
rate (                          # without time-dep covariates
    xa(0,1) = COHO2,COHO3,W,
) = 1;
rate (                          # using episode splitting
    xa(0,1) = COHO2,COHO3,W,MARR,
) = 1;
rate (                          # using standard approach
    xa(0,1) = COHO2,COHO3,W,MARR1,
) = 1;
```

a type 5 variable, meaning that this variable will be re-evaluated when TDA estimates a Cox model which contains this variable. Then follow three `rate` commands.

1. The first one re-estimates the model discussed in Example **6.17.7.1**-1. Although model estimation now uses split episodes, one should get identical estimation results.

2. The second `rate` command re-estimates the model discussed in Example **6.17.7.2**-1, using now the method of episode splitting for the time-

dependent variable `MARR`.

3. The third `rate` command estimates the same model, but uses the conventional method for time-dependent covariates in Cox models; the model specification contains now `MARR1` instead of `MARR`. Since `MARR1` is a type 5 variable, it is re-evaluated whenever the risk set must be updated. Consequently, the estimation procedure is much more time consuming.

6.17.7.4 The Proportionality Assumption

A basic feature of the Cox model is that transition rates for different values of covariates are proportional. If, for instance, we compare the transition rates $r_1(t)$ and $r_2(t)$, corresponding to the values of the ith covariate, A_i and A'_i, then

$$\frac{r_1(t)}{r_2(t)} = \exp\{(A_i - A'_i)\,\alpha_i\}$$

Models having this feature are called proportional transition rate models. This is, of course, not specific for the Cox model but also implied in at least some versions of most of the parametric models discussed in 6.17.3. This section discusses three methods to check whether a sample of episode data allows for this proportionality assumption.

Graphical Methods. If there are only a few categorical covariates in a model, it is possible to perform simple graphical checks. Assume that this is the case, and that A and A' are two different vectors with covariate values. The sample can then be split into two groups, and one can calculate nonparametric Kaplan-Meier estimates of the survivor functions, say $G(t)$ and $G'(t)$, for both of these groups. The proportionality assumption implies that

$$\frac{\log(G(t))}{\log(G'(t))} = \exp\{(A - A')\,\alpha\}$$

Or, taking logarithms once more, this becomes

$$\log\{-\log(G(t))\} = \log\{-\log(G'(t))\} + (A - A')\,\alpha \qquad (1)$$

Therefore, a plot of the logarithms of the survivor functions may be used to check if the proportionality assumption is (nearly) admissible. Similar ideas can be used to check parametric model assumptions, see Wu [1990].

Testing Time-dependence. Another method to check the proportionality assumption is to look at possible interaction effects between covariates and process time. Assuming that the ith component of the covariate vector A shall be tested, a model like the following could be

Box 1 Command file `pl6.cf`

```
nvar(
    dfile = rrdat.1,   # data file

    ID    [3.0] = c1,  # identification number
    SN    [2.0] = c2,  # spell number
    TS    [3.0] = c3,  # starting time
    TF    [3.0] = c4,  # ending time
    SEX   [2.0] = c5,  # sex (1 men, 2 women)
    TI    [3.0] = c6,  # interview date
    TB    [3.0] = c7,  # birth date
    TE    [3.0] = c8,  # entry into labor market
    TMAR  [3.0] = c9,  # marriage date (0 if no marriage)
    PRES  [3.0] = c10, # prestige of current job
    PRESN [3.0] = c11, # prestige of next job
    EDU   [2.0] = c12, # highest educational attainment

    COHO1 = ge(TB,348) & le(TB,384), # birth cohort 1
    COHO2 = ge(TB,468) & le(TB,504), # birth cohort 2
    COHO3 = ge(TB,588) & le(TB,624), # birth cohort 3
    W     = SEX[2],                  # women = 1

    DES [1.0] = if eq(TF,TI) then 0 else 1,  # destination state
    DUR [3.0] = TF - TS + 1,
);
edef(
    ts = 0,     # starting time
    tf = DUR,   # ending time
    org = 0,    # origin state
    des = DES,  # destination state
    WTEST = W * (log(time) - 4.22),
);
rate (
    xa(0,1) = COHO2,COHO3,W,WTEST,

) = 1;
```

set up and estimated.

$$r(t) = h(t)\,\exp\{A\,\alpha + A_i\,(\log(\texttt{time}) - M)\,\alpha_i'\} \tag{2}$$

The constant M is added to facilitate the estimation. It should be the logarithm of the mean (or median) duration of the sample episodes used for the estimation. If the added variable gives a significant coefficient, i.e., if there are significant interaction effects, the proportionality assumption for this covariate is probably not in order.

Box 2 Part of standard output from command file `pl6.cf`

```
Maximum of log likelihood: -2562.9
Norm of final gradient vector: 6.79524e-07
Last absolute change of function value: 1.71305e-09
Last relative change in parameters: 0.00104722

Idx SN Org Des MT Variable   Coeff    Error    C/Error  Signif
----------------------------------------------------------------
  1  1   0   1  A COHO2     0.3816   0.1133     3.3682  0.9992
  2  1   0   1  A COHO3     0.4381   0.1180     3.7136  0.9998
  3  1   0   1  A W         0.5840   0.1215     4.8074  1.0000
  4  1   0   1  A WTEST     0.2196   0.0974     2.2541  0.9758

Log likelihood (starting values): -2584.5701
Log likelihood (final estimates): -2562.9031
```

Example 1 To illustrate this procedure, we check variable `W` (sex=2, women) that was used in Example 1 in **6.17.7.1**. The command file is now `pl6.cf` shown in Box 1. It is mostly identical to command file `pl1.cf`, but we have added the time-dependent covariate `WTEST`. Simply following (4), we use the definition

```
WTEST = W * (log(time) - 4.22),
```

where 4.22 is approximately the logarithm of the mean duration in our example data set. Estimation results are shown in Box 2. Obviously, variable `W` creates some conflicts with the proportionality assumption.

A Goodness-of-Fit Test. As proposed by Moreau et al. [1985], the idea of checking for interactions between covariates and the process time can be generalized for a global goodness-of-fit test for the Cox model. We briefly describe this proposal and its implementation in TDA. Is is assumed that all episodes start at the origin of the process time, i.e. have starting time zero. The process time axis is divided into intervals according to

$$0 \leq \tau_1 < \tau_2 < \ldots < \tau_q, \quad \tau_0 = 0, \quad \tau_{q+1} = \infty$$

The lth time interval is defined as

$$I_l = \{t \mid \tau_l \leq t < \tau_{l+1}\} \qquad l = 1, \ldots, q$$

Moreau et al. [1985] propose to look at a general model formulation that allows possibly different covariate effects in the time intervals. In the

single transition case, the model can be formulated as

$$r(t) = h(t)\, \exp\{A(\alpha + \gamma_l)\} \qquad \text{if} \quad t \in I_l \tag{3}$$

In the lth time interval, the coefficients associated with the covariate vector A are given by $\alpha + \gamma_l$. If the proportionality assumption holds, then all γ_l vectors should be zero. Since one of these vectors is obviously redundant, we set $\gamma_1 = 0$, and the null hypothesis becomes

$$\gamma_2 = \gamma_3 = \ldots = \gamma_q = 0 \tag{4}$$

For testing this hypothesis, Moreau et al. [1985] propose a score test. The test statistic, say S, is defined by

$$S = U'\, V^{-1}\, U$$

where U is a vector of first derivatives of the log-likelihood of model (3), evaluated under the null hypothesis (4) and with maximum partial likelihood estimates of the vector α. And V is the observed information matrix, i.e. minus the matrix of second derivatives of the log-likelihood of (3), evaluated the same way. Under the null hypothesis, the statistic S is asymptotically χ^2 distributed with degrees of freedom equal to the number of parameters of model (3), that is $(q-1)\,p$ where p is the dimension of α.

As shown by Moreau et al. [1985], the calculation of S can be greatly simplified. It is possible to write the test statistic as

$$S = \sum_{l=1}^{q} U_l'\, V_l^{-1}\, U_l$$

with U_l and V_l restricted to the lth time interval. The elements of U_l and V_l $(s, s_1, s_2 = 1, \ldots, p)$ may then be written as

$$U_{l,s} = \sum_{i \in \mathcal{E}_l} A_{i,s} - \frac{\sum_{j \in \mathcal{R}_{i,l}} \exp(A_j\, \hat{\alpha})\; A_{j,s}}{\sum_{j \in \mathcal{R}_{i,l}} \exp(A_j\, \hat{\alpha})}$$

$$V_{l,s_1 s_2} = \sum_{i \in \mathcal{E}_l} \frac{\sum_{j \in \mathcal{R}_{i,l}} \exp(A_j\, \hat{\alpha})\; A_{j,s_1} A_{j,s_2}}{\sum_{j \in \mathcal{R}_{i,l}} \exp(A_j\, \hat{\alpha})} - \frac{\sum_{j \in \mathcal{R}_{i,l}} \exp(A_j\, \hat{\alpha})\; A_{j,s_1}}{\sum_{j \in \mathcal{R}_{i,l}} \exp(A_j\, \hat{\alpha})} \frac{\sum_{j \in \mathcal{R}_{i,l}} \exp(A_j\, \hat{\alpha})\; A_{j,s_2}}{\sum_{j \in \mathcal{R}_{i,l}} \exp(A_j\, \hat{\alpha})}$$

These expressions are analogous to the derivatives of (5) given above, restricted to single time periods. $\mathcal{E}_l$ is the set of episodes with events in the lth time interval; $\mathcal{R}_{i,l}$ is the risk set at the ending time of the ith episode in $\mathcal{E}_l$, and $\hat{\alpha}$ is the maximum partial likelihood estimate of α under the null hypothesis. As proposed by Moreau et al. [1985], the formulation in (5) accounts for tied ending times by using Breslow's approximation.

A generalization to the case of multiple transitions is simple since the log-likelihood factors into separate components for each transition. Therefore, a separate test statistics can be calculated for each transition, say (j, k), taking into account all episodes with origin state j and regarding all episodes as censored that do not end with destination state k.

To request a calculation of the test statistics, one has to add a definition of time periods to the specification of a Cox model. The test statistics is printed, then, into the standard output. In addition to the test statistics and associated degrees of freedom, TDA prints the corresponding value of the χ^2 distribution function.

If a time interval is empty, meaning that there are no episodes with ending times in this interval, it is not used for the calculation of the test statistic and not counted in the degrees of freedom. Sometimes, mainly depending on the definition of time intervals, one or more of the V_l matrices is rank-deficient and cannot be inverted. The test statistic is not calculated, then, but some information about the underlying problem is given, on request, in the protocol file.

Example 2 To illustrate the test procedure, we first reproduce an example given by Moreau et al. [1985]. The data, `rrdat.5`, are survival times of gastric carcinoma patients who are given two different therapies. The command file, `pl7.cf`, is shown in Box 3. Part of the standard output is shown in Box 4. The test statistic is significant, indicating that the effect of the two therapies (groups) is not proportional.

Example 3 For a second illustration, we add the parameter

```
tp = 0 (12) 96,
```

to command files `pl1.cf` `pl1m.cf`, see the first two examples in **6.17.7.1**. The new command files are `pl8.cf` and `pl8m.cf`, not shown, but contained in the example archive. Box 5 shows the test statistics created by these command files. In the first example, without alternative destination states, the test statistic is 37.108 with 24 degrees of freedom.

Box 3 Command file `pl7.cf`

```
nvar(
    dfile = rrdat.5,    # data file

    TF  = c1,           # ending time = duration
    DES = c2,           # destination state (= status)
    G1  = c3[1],        # definition of two groups
    G2  = c3[2],
);
edef(
    ts = 0,             # starting time = 0
    tf = TF,            # ending time = duration
    org = 0,            # origin state = 0
    des = DES,          # destination state (= status)
);
rate(
    xa(0,1) = G1,               # included: group 1
    tp = 0,170.1,354.1,535.1,   # time periods
) = 1;
```

Box 4 Part of standard output from command file `pl7.cf`

```
Maximum of log likelihood: -282.744
Norm of final gradient vector: 9.13651e-08
Last absolute change of function value: 3.13773e-07
Last relative change in parameters: 0.0116559

Idx SN Org Des MT Variable   Coeff   Error    C/Error  Signif
-------------------------------------------------------------
  1  1   0   1  A G1       -0.2666  0.2332    -1.1428  0.7469

Log likelihood (starting values):  -283.3952
Log likelihood (final estimates):  -282.7441
Global Goodness-of-fit.

SN  Org  Des        TStat    DF   Signif
---------------------------------------
 1    0    1      10.2351     3   0.9833
```

As expected, it shows a significant deviation from the proportionality assumption. Again, this is mainly due to the `W` variable. Omitting this variable from the model gives a test statistic of 21.04 with 16 degrees of freedom which is no longer significant.

Box 5 Test statistics created by command files `pl8.cf` and `pl8m.cf`

`pl8.cf (corresponding to pl1.cf)`

SN	Org	Des	TStat	DF	Signif
1	0	1	37.1084	24	0.9573

`pl8m.cf (corresponding to pl1m.cf)`

SN	Org	Des	TStat	DF	Signif
1	0	1	15.9066	24	0.1086
1	0	2	25.5454	24	0.6234
1	0	3	38.5791	24	0.9698

6.17.7.5 Stratification

If the proportionality assumption is not acceptable, it is sometimes sensible to estimate a stratified model. This is possible if the covariates that cause conflicts with the proportionality assumption are categorical. The whole sample can then be split into groups (strata), one group for each of the possible combinations of categories. In the simplest case, with only a single dummy variable, there will be only two groups.

Let $\mathcal{G}$ denote the set of groups. The model should be specified such that the baseline rate can be different for each group. There are two possibilities. One possibility is to define a different model for each group; then not only the baseline rates, but also the covariate effects can vary across groups.

This approach requires fairly large samples. When data sets are relatively small, or when there are no significant interaction effects, it can be sensible to build on the assumption that covariate effects are the same in all groups. (See Kalbfleisch and Prentice [1980], p. 87; Lawless [1982], p. 365; Blossfeld et al. [1989].) Based on this assumption, a model for the single transition case can be written as

$$r_g(t) = h_g(t)\, \exp(A\,\alpha) \qquad g \in \mathcal{G}$$

To estimate this model, the partial likelihood has to be calculated as the product of the group-specific likelihoods. For the multiple transition case, this may be written as

$$L^p = \prod_{j\in\mathcal{O}} \prod_{k\in\mathcal{D}_j} \prod_{g\in\mathcal{G}_j} \prod_{i\in\mathcal{E}_{jk,g}} \frac{\exp\left(A^{(jk)}(t_i)\,\alpha^{(jk)}\right)}{\sum_{l\in\mathcal{R}_{j,g}(t_i)} \exp\left(A^{(jk)}(t_l)\,\alpha^{(jk)}\right)}$$

with $\mathcal{G}_j$ denoting the set of groups of episodes with origin state j. $\mathcal{E}_{jk,g}$ is the set of all not censored episodes having origin state j and destination state k *and* belong to group g. Accordingly, $\mathcal{R}_{j,g}(t)$ is the risk set at time t containing all episodes with origin state j, starting time less than t, ending time equal to or greater than t, and belonging to group g. Box 1 in 6.17.7.1 shows how this is implemented in TDA's partial likelihood algorithm.

Example 1 To illustrate stratified estimation, we continue with the first example in 6.17.7.1 and use the variable `SEX` for stratification. Box

Box 1 Command file `pl9.cf` for stratified estimation

```
nvar(
    dfile = rrdat.1,   # data file

    ID    [3.0] = c1,  # identification number
    SN    [2.0] = c2,  # spell number
    TS    [3.0] = c3,  # starting time
    TF    [3.0] = c4,  # ending time
    SEX   [2.0] = c5,  # sex (1 men, 2 women)
    TI    [3.0] = c6,  # interview date
    TB    [3.0] = c7,  # birth date
    TE    [3.0] = c8,  # entry into labor market
    TMAR  [3.0] = c9,  # marriage date (0 if no marriage)
    PRES  [3.0] = c10, # prestige of current job
    PRESN [3.0] = c11, # prestige of next job
    EDU   [2.0] = c12, # highest educational attainment

    COHO1 = ge(TB,348) & le(TB,384), # birth cohort 1
    COHO2 = ge(TB,468) & le(TB,504), # birth cohort 2
    COHO3 = ge(TB,588) & le(TB,624), # birth cohort 3
    G1    = SEX[1],                  # group1 (men)
    G2    = SEX[2],                  # group2 (women)

    DES [1.0] = if eq(TF,TI) then 0 else 1,  # destination state
    DUR [3.0] = TF - TS + 1,
);
edef(
    ts = 0,     # starting time
    tf = DUR,   # ending time
    org = 0,    # origin state
    des = DES,  # destination state
);
rate (
    xa(0,1) = COHO2,COHO3,
    grp = G1,G2,            # groups for stratification
) = 1;
```

1 shows the command file `pl9.cf`. The additional variables `G1` and `G2` define two groups, men and women.[1] To request stratified estimation, one has to use the parameter

```
grp = G1,G2,...,
```

in the `rate` command. TDA then assumes that the right-hand side is

[1] In TDA, groups are always defined by indicator variables. Nonzero values indicate that a case belongs to the respective group.

Box 2 Part of standard output from command file `pl9.cf`

```
Model: Cox (partial likelihood)
Stratified with 2 groups.

Maximum likelihood estimation.
Algorithm 5: Newton (I)

Convergence reached in 4 iterations.
Number of function evaluations: 5 (5,5)

Maximum of log likelihood: -2250.58
Norm of final gradient vector: 8.82782e-09
Last absolute change of function value: 1.43084e-11
Last relative change in parameters: 4.27658e-05

Idx SN Org Des MT Variable   Coeff    Error    C/Error  Signif
-----------------------------------------------------------------
  1  1   0   1  A COH02     0.3840   0.1133     3.3878  0.9993
  2  1   0   1  A COH03     0.4428   0.1183     3.7433  0.9998

Log likelihood (starting values): -2259.5669
Log likelihood (final estimates): -2250.5832
```

a list of indicator variables specifying groups and performs the stratified estimation procedure based on these groups. Estimation results are shown in Box 2. As is seen the cohort effects have not changed very much. This is consistent with that both variables do not give significant results when tested for proportionality.

6.17.7.6 The Baseline Rate

The partial likelihood method provides estimates of the parameters of a Cox model but no direct estimate of the underlying baseline rate. Clearly, this would be useful in order to provide some information about the structure of time-dependence and could be used, for instance, to check whether also a fully parametric model could be appropriate.

There are different proposals for estimators of the baseline rate. TDA's approach is based on a proposal by Breslow [1974], discussed also in Blossfeld et al. [1989]. To explain the method, we begin with assuming a single transition where all episodes have starting time zero. Then, in the usual notation, $\mathcal{E}$ denotes the set of, say q, not censored episodes, and the ordered ending times may be given by

$$\tau_1 < \tau_2 < \ldots < \tau_q$$

Let E_i be the number of events at τ_i, let $\mathcal{R}_i$ be the risk set at τ_i. Furthermore, let $\hat{\alpha}$ be the partial likelihood estimate of the model parameters. Then, defining $\tau_0 = 0$, we may regard

$$\hat{h}_i^b = \frac{E_i}{(\tau_i - \tau_{i-1}) \sum_{l \in \mathcal{R}_i} \exp(A_l\, \hat{\alpha})} \qquad i = 1, \ldots, q$$

as an estimate of the baseline rate of the model. It is a constant during each interval $(\tau_{i-1}, \tau_i]$, resulting in a step function with steps at the points in time where at least one event occurs. This step function can be integrated to provide an estimate of the cumulative baseline rate.

$$\hat{H}^b(t) = \sum_{l=1}^{i} (\tau_l - \tau_{l-1})\, \hat{h}_l + (t - \tau_i)\, \hat{h}_{i+1} \qquad \tau_i < t \leq \tau_{i+1}$$

The resulting estimate of the baseline survivor function is

$$\hat{G}^b(t) = \exp\left\{-\hat{H}^b(t)\right\}$$

Finally, because of the assumption of proportional effects, the cumulative transition rate for an arbitrary covariate vector A can be estimated by

$$\hat{H}(t) = \hat{H}^b(t)\, \exp\left(A\hat{\alpha}\right) \tag{1}$$

Box 1 `rate` command in command file `pl1r.cf`

```
rate (
    xa(0,1) = COHO2,COHO3,W,
    prate (COHO3=1) = r.dat,
    prate (COHO3=1,W=1) = r.dat,
) = 1;
```

and the corresponding survivor function estimate is

$$\hat{G}(t) = \exp\left\{-\hat{H}^b(t)\,\exp\left(A\hat{\alpha}\right)\right\} = \hat{G}^b(t)^{\exp(A\hat{\alpha})} \tag{2}$$

TDA uses these formulas to provide estimates of the cumulative baseline transition rate and the corresponding baseline survivor function. A generalization to the case of multiple transitions is done analogously to a product-limit estimation for several transitions: each transition is treated separately. For instance, with respect to the transition (j, k), the procedure outlined above is applied to all episodes with origin state j; all episodes not ending in destination state k are regarded as censored. The resulting estimates are then pseudosurvivor functions and transition-specific cumulative rates, respectively.

To request an estimation of cumulated (baseline) rates, one can use the `prate` option described in **6.17.1.4**. Note that TDA calculates *cumulated* rates. To get estimates of the rates, one should first smooth the cumulated rates and then use numerical differentiation.

Example 1 To illustrate the calculation of cumulative baseline rates, we use command file `pl1r.cf` (not shown). It is basically identical with command file `pl1.cf` (Example 1 in **6.17.7.1**). Two `prate` parameters have been added to the `rate` command as shown in Box 1. The first `prate` parameter calculates an estimate of the cumulated rate with the covariate values `COHO2=0`, `COHO3=1`, and `W=0`. The table is written into the output file `r.dat`. The second `prate` parameter creates a second table, written to the same output file, for covariate values `COHO2=0`, `COHO3=1`, and `W=1`. Part of the first table in output file `r.dat` is shown in Box 2.

Box 2 Part of output file `r.dat`

```
# Cox Model. Baseline Rate Calculation.

# Idx SN Org Des MT Variable      Coeff  Covariate
# ------------------------------------------------
#   1  1   0   1  A COHO2         0.3797     0.0000
#   2  1   0   1  A COHO3         0.4585     1.0000
#   3  1   0   1  A W             0.4078     0.0000

# Transition: SN 1  Org 0  Des 1

# ID      Time    Events    Censored    Risk Set    Surv.F.    Cum.Rate
# ---------------------------------------------------------------------
   0    0.0000    0.0000      0.0000    600.0000     1.0000      0.0000
   0    2.0000    2.0000      0.0000    600.0000     0.9967      0.0033
   0    3.0000    5.0000      1.0000    597.0000     0.9885      0.0115
   0    4.0000    9.0000      2.0000    590.0000     0.9737      0.0266
   0    5.0000    3.0000      0.0000    581.0000     0.9688      0.0317
   0    6.0000   10.0000      1.0000    577.0000     0.9523      0.0488
```

6.18 Regression Models for Events

This chapter describes a simple approach to model conditions for the occurrence of events, based on sequence data (see **3.4**). The sections are:

6.18.1 A Simple Modeling Approach

We assume a K-dimensional sequence data structure (see **3.4**). (y_{ik}) denotes the sequence of the ith individual (sample member) in the kth sequence data structure. We are interested in a model that shows how the occurrence of events depends on certain conditions. Such a model can be constructed for each type of event. To simplify the approach, we focus on just one event, say $[k, s_1, s_2]$, that is, a transition from state s_1 to state s_2 in the kth state space (sequence data structure). In the following, this will be called the target event. We can then construct an indicator variable

$$Z_{it} = \begin{cases} 1 & \text{if } y_{i,t-1} = s_1 \text{ and } y_{i,t} = s_2 \\ 0 & \text{otherwise} \end{cases}$$

Z_{it} is 1 if, for individual i, the target event occurs at t. In general, we will say that an event occurs *at* t if the corresponding transition occurs between $t-1$ and t. This implies that we can observe events only in a time interval $[t_a, t_b]$ with

$$T_k^{\min} < t_a \leq t_b \leq T_k^{\max}$$

($T_k^{\min}$ and $T_k^{\max}$ define the time axis for the $k$the sequence data structure, see **3.4.2**.) Of course, to set up a model we are free to choose any subset $\mathcal{T}_m \subset [t_a, t_b]$. Note that $\mathcal{T}_m$ can be a calendar time axis or a process time axis.

Focus of the model is the conditional probability for the occurrence of the target event. For a time point $t \in \mathcal{T}_m$, a general parametric approach can be written as

$$\Pr(Z_{it} = 1 \,|\, x_{it}) = g(t, x_t, \theta_t) \tag{1}$$

$g(.)$ is some function, constrained to $0 \leq g(.) \leq 1$, depending on time t, a time-dependent vector of covariates, x_t, and a parameter vector, θ_t. Model specification will then consist of two parts. First, to choose a specification of the function $g(.)$. The most simple choice would be a logistic regression specification. Second, and in a sense more important, is then a specification of the covariates. This will be discussed below.

To estimate the model, one needs the concept of a risk set. Consistent with our definition of events, the risk set $\mathcal{R}_t$ for the target event $[k, s_1, s_2]$ contains all sample members who are in state s_1 at time $t-1$ and have some valid state at t. Using z_{it} to denote realized values of Z_{it}, the log-likelihood of model (1) can then be written as

$$\ell_t = \sum_{i \in \mathcal{R}_t} z_{it} g(t, x_t, \theta_t) + (1 - z_{it})(1 - g(t, x_t, \theta_t)) \tag{2}$$

So far, we have separate models for each time point $t \in \mathcal{T}_m$. To arrive at a single model for the whole time axis $\mathcal{T}_m$, we can simply consider and estimate the models simultaneously. The log-likelihood then becomes

$$\ell = \sum_{t \in \mathcal{T}_m} \sum_{i \in \mathcal{R}_t} z_{it} g(t, x_t, \theta_t) + (1 - z_{it})(1 - g(t, x_t, \theta_t)) \tag{3}$$

This allows to consider constraints for the time-specific parameter vectors θ_t and should be seen as an essential part of the model specification.

Models of this kind can be easily estimated if one uses some standard specification, e.g., a logit or probit model. The only difficulty consists in the construction of sensible covariates. A causal modeling approach can begin with the idea that, what happens at some time point t, can, in principle, depend on any facts earlier than t. Here, we only shortly indicate some different kinds of covariates. How to construct these covariates will be discussed in **6.18.3**.

1. Without any covariates, (3) will give an estimate of a mean probability for the occurrence of the target event during the time axis $\mathcal{T}_m$. In many applications one would like to consider also period effects. This can be achieved by constructing dummy variables for each time point and then estimating a single coefficient for each of these variables.

2. Also time-independent covariates can be specified as an identical part in each of the covariate vectors x_{it}, and it would normally suffice to estimate a single effect for each of these variables.

3. More interesting are events in the past history of the process. Given a time point t, one should consider all possibly important events which occurred earlier than t. Let $\mathcal{E}$ denote the set of possibly important events. A simple way would be to define, for each $e \in \mathcal{E}$, a variable $e_{i,t}$ that simply counts how often the event e occurred before t. Depending on the kind of event and its frequency of occurrence, these variables can be used directly as part of x_{it}, or transformed into suitable dummy variables.

4. In addition, one can consider the duration between the occurrence of an event, say at t', and the current process time t. This could make sense if the event should have happened for each individual before entering the time axis of the model, $\mathcal{T}_m$. The most important example is birth, and the duration until t is simply age.

5. However, duration between occurrence of an event and current process time t is only defined if the event has, in fact, occurred in the past of t. In general, events in $\mathcal{E}$ are contingent and may, or may not, occur. It seems preferable, then, to use the concept of an *effect shape* (instead of duration).[1] To provide a formulation, we will use (e, τ) to denote events; $e \in \mathcal{E}$ is the type of the event, and τ is the time when it occurred.[2] For each event (e, τ), we can define an effect shape as a function of time in the following way:

$$h(t; e, \tau, \theta_e) = \begin{cases} h_e(t - \tau; \theta_e) & \text{if } t > \tau \\ 0 & \text{otherwise} \end{cases}$$

The idea is, given an event of type e occurred at $\tau < t$, its possible impact on the occurrence of the target event at t is given by the value of its effect shape at t. In this way, one can consider the combined impact of any number of events that happened before t; and, of course, this impact dynamically changes while t runs through the time axis, $\mathcal{T}_m$.

Unfortunately, we most often cannot justify a priori assumptions about the functional form, $h_e(.)$, of an effect shape. Instead, we would like to get some evidence about effect shapes from the data. However, since our time axis is discrete, we can simply consider, for each type of event $e \in \mathcal{E}$, a set of $d + 1$ dummy variables $e_{i,t,(j)}$, $j = 0, \ldots, d$, $d \geq 0$; using the definition

$$e_{i,t,(j)} = \begin{cases} 1 & \text{if there is an event } (e, \tau) \text{ and } t = \tau + j \\ 0 & \text{otherwise} \end{cases}$$

The effects of these event-specific dummy variables can then provide

[1] For a discussion of this concept, see Blossfeld and Rohwer [1995]. In a certain sense, the approach can be viewed as a generalization of conventional transition rate modeling based on the duration of episodes. For some background discussion, see Mayer and Huinink [1990], Rohwer [1996].

[2] Notice that we should always distinguish between kinds of events, here given by the set $\mathcal{E}$, and "'concrete"' events that actually occurred. For a discussion of this distinction see, e.g., Galton [1984], ch. 2.

some evidence about the effect shapes.[3]

How to construct these different kinds of covariates, based on a K-dimensional sequence data structure, will be discussed in **6.18.3**.

[3]For an application of this approach to estimate the effect shape of pregnancies for the target event marriage (in consensual unions), see Blossfeld et al. [1996].

6.18.2 Investigating Events

Before trying to construct regression models for events, one should investigate the events actually occurring in a given sample of sequence data. TDA provides two simple commands. The first command is `seqev` with syntax shown in the following box.

```
seqev (
    sn=...,      number of sequence data structure, def. 1
    sel=...,     expression for sequence selection
) = fname;
```

All parameters are optional and have their usual meaning. If no file name is given on the right-hand side, the command prints information about the events in the selected sequence data structure into the standard output; otherwise into the file specified on the right-hand side. The information about events is a table with three columns and a separate line for each type of event. The first two columns indicate the type of event (origin and destination state), and the third column shows how often this event occurs in the selected sequence data structure.

Example 1 To illustrate the `seqev` command, we use the data file `seq.d4` shown in Box 1. There are two sequences for each of three individuals, both defined on the same time axis, $t = 0, 1, \ldots, 5$; and in addition, two variables, `V1` and `V2`, which will be used for some illustrations in **6.18.3**. Box 2 shows the command file, `seq8.cf`, that was used to read the data, define two sequence data structures, and count the number of events in the first one. Part of the standard output is shown in Box 3. There are two types of events. The first one, $(1, 2)$, occurs four times, the second one, $(2, 1)$ occurs three times.

Box 1 Sequence data file `seq.d4`

```
ID  0 1 2 3 4 5  0 1 2 3 4 5  V1  V2
--  -----------  -----------  --  --
 1  1 1 2 2 1 1  1 3 3 1 1 3   1 -10
 2  1 2 1 1 1 2  1 1 1 3 3 3   2 -15
 3  2 2 1 1 2 2  1 1 3 1 3 3   2 -20
```

Box 2 Command file `seq8.cf`

```
nvar(
    dfile = seq.d4,
    ID     = c1,
    Y{0,5} = c2,
    S{0,5} = c8,
    V1     = c14,
    V2     = c15,
);
seqdef(sn=1) = Y0,,Y5;
seqdef(sn=2) = S0,,S5;
seq;                info about current sequence data
seqev;              count events in first sequence data structure
```

Box 3 Part of standard output from `seq8.cf`

```
Sequence          State       Time axis       Number
Structure Type  Variables  Minimum  Maximum  of States  States
-------------------------------------------------------------
     1      1         6          0        5         2    1 2
     2      1         6          0        5         2    1 3

Range of common time axis: 0 to 5.

Counting events.
Using sequence data structure 1.

Event type    Number
--------------------
   1    2          4
   2    1          3
```

The `seqevd` command. The second command is `seqevd` with syntax shown in the following box.

```
seqevd (
    sn=...,        number of sequence data structure, def. 1
    sel=...,       expression for sequence selection
    dtda=...,      request TDA description file
) = fname;
```

A file name must be given on the right-hand side, all other parameters are optional and have their usual meaning. The command creates the specified output file and writes into this output file a table showing, for each time point, the number of events occurring at that time point. We say that an event occurs at t if there is a corresponding transition between $t-1$ and t.

Box 4 Illustration of `seqevd` command

```
Output file from command: seqevd(dtda=t)=d

Time Cases EV1_2 EV2_1  NEV
---------------------------
  1      3    1    0      1
  2      3    1    2      3
  3      3    0    0      0
  4      3    1    1      2
  5      3    1    0      1

Output file created by dtda=t

nvar(
  dfile = d,
  noc = 5,
  TIME <5>[6.0] = c1 ,  # time
  NCAS <5>[6.0] = c2 ,  # number of cases
  EV1_2 <5>[4.0] = c3 , # number of events (1,2)
  EV2_1 <5>[4.0] = c4 , # number of events (2,1)
  NEV <5>[6.0] = c5 ,   # total number of events
);
```

Example 2 To illustrate the structure of the output file, we add the command

```
seqevd (dtda = t) = d;
```

to command file `seq8.cf`. Box 4 shows the resulting output file. The column labelled `Cases` contains the number of sequences having a valid state at $t-1$ and t. The additional output file created by the `dtda` parameter provides an `nvar` command to read the data file. This is useful, for instance, if one wishes to create plots of the frequency distributions of events.

6.18.3 Data Generation

This section describes how to create a data file which can subsequently be used for estimating regression models for events, following the approach discussed in **6.18.1**. Having created such a data file, model estimation can use any of the models for binary response variables described in **6.12**. The command to create a data file is `seqmd` with syntax shown in the following box.

```
seqmd (
    ev=...,        definition of target event
    sel=...,       expression for sequence selection
    tp=...,        definition of time axis
    v=...,         time-independent covariates
    xe=...,        event-specific covariates
    dtda=...,      request TDA description file
) = fname;
```

With the exception of `ev`, all parameters are optional. The `ev` parameter must be used to define a target event. The syntax is

`ev =` $[k, s_1, s_2]$`,`

where k is the number of a sequence data structure and s_1 and s_2 are the origin and destination state of the event.

1. The parameter

`sel = expression,`

can be used to select a subset of sequences. The command then uses only sequences (data matrix cases) where `expression` evaluates to a nonzero value.

2. The command uses a time axis $[t_a, t_b]$ with $t_a = T_k^{\min} + 1$ and $t_b = T_k^{\max}$ by default. Alternatively, one can specify a time axis with the `tp` parameter. The syntax is

$\text{tp} = t_1, t_2, \ldots, t_n$ or $\text{tp} = t_1\,(d)\,t_2$

Box 1 Illustration of `seqmd` command

```
Command: seqmd (ev=[1,1,2]);

Selected target event: [1,1,2]
Using sequence data structure 1.
Selected time axis for events: 1 -- 5

Time  Risk Set  Events
----------------------
   1         2       1
   2         1       1
   3         2       0
   4         2       1
   5         2       1
```

or a mixture of both expressions. Assuming $d > 0$, the second expression is expanded into the sequence $t_1 + id$ $(i = 0, 1, 2, \ldots)$, as long as the result is less than or equal to t_2. Note that a time axis defined with the `tp` parameter must be a subset of the default time axis. As in **6.18.1**, we will use $\mathcal{T}_m$ to denote the selected time axis.

3. If the `seqmd` command is used without providing a file name on the right-hand side, the command only displays a table showing the risk set and the number of events for the specified time axis. To illustrate, we use the data file `seq.d4` (see **6.18.2**) and add the following command to command file `seq8.cf`:

```
seqmd(ev = [1,1,2]);
```

This command specifies the target event $[1, 2]$ in sequence data structure 1 and uses the default time axis. The resulting table is shown in Box 2. In this example, the risk set $\mathcal{R}_t$ contains all individuals (sample members) being in state 1 at $t - 1$ and having a valid state at t.

4. The command only creates a data file if a name is given on the right-hand side. The data file will contain $n = \Sigma n_i$ records where n_i is the number of time points the individual i belongs to the risk set. Without any additional parameters specifying covariates, the data file will contain the following variables:

- An `ID` variable which is equal to the case number of the sequence in the data matrix.
- The current time point, t.

Box 2 Illustration of `seqmd` command

```
Command: seqmd(ev = [1,1,2]) = out.d;
Output file:
                period
ID  Time  Z     dummies
-----------------------
 1    1   0    1 0 0 0 0
 1    2   1    0 1 0 0 0
 1    5   0    0 0 0 0 1
 2    1   1    1 0 0 0 0
 2    3   0    0 0 1 0 0
 2    4   0    0 0 0 1 0
 2    5   1    0 0 0 0 1
 3    3   0    0 0 1 0 0
 3    4   1    0 0 0 1 0
```

Box 3 Illustration of `seqmd` command

```
Command: seqmd(ev = [1,1,2], v = V1,V2) = out.d;
Output file:
                period
ID  Time  Z     dummies    V1   V2
-------------------------------
 1    1   0    1 0 0 0 0    1  -10
 1    2   1    0 1 0 0 0    1  -10
 1    5   0    0 0 0 0 1    1  -10
 2    1   1    1 0 0 0 0    2  -15
 2    3   0    0 0 1 0 0    2  -15
 2    4   0    0 0 0 1 0    2  -15
 2    5   1    0 0 0 0 1    2  -15
 3    3   0    0 0 1 0 0    2  -20
 3    4   1    0 0 0 1 0    2  -20
```

- An indicator variable, Z_{it}, being 1 if the target event occurs for individual i at time t, and otherwise zero.
- For each $j \in \mathcal{T}_m$, a dummy variable $P_{it,(j)}$, being 1 if $j = t$ and zero otherwise. These dummy variables can be used to model period effects.

To illustrate this option, we add the command

```
seqmd(ev = [1,1,2]) = out.d;
```

to command file `seq8.cf`. The new command file is `seq8a.cf`. The output file, `out.d`, is shown in Box 3.

5. The `v` parameter can be used to add time-independent covariates to the output file. The syntax is

```
v = varlist,
```

where `varlist` is a list of variables which must be available in the currently defined TDA data matrix. To illustrate this option, we use the two variables, `V1` and `V2`, from the data file `seq.d4` (see **6.18.2**). The command is

```
seqmd(ev = [1,1,2], v=V1,V2) = out.d;
```

The new command file is `seq8b.cf`. The resulting output file is shown in Box 3. For writing these variables, TDA uses their associated print formats. In our example, it is a free format.

6. We have finally to discuss how to create event-specific covariates. This can be done with the `xe` parameter. The syntax is

```
xe = [....],[....],...,
```

where each pair of square brackets specifies one set of covariates, related to one type of event. There are two possibilities to define the event-specific covariates.

7. As a first option, one can use variables contained in TDA's currently defined data matrix. For example, `xe=[V]`, where `V` is the name of a variable. It is assumed, then, that `V` records the dates of an event. The `xe` parameter then creates a covariate, named `V_D`, containing the duration between the date in `V` and the current process time, t.

To illustrate this option, we use the variables `V1` and `V2` in data file `seq.d4` and add the command

```
seqmd(ev = [1,1,2], v=V1,V2, xe=[V1],[V2]) = out.d;
```

The new command file is `seq8c.cf`. The resulting output file is shown in Box 4. Note that the values of these covariates will be negative if the date is greater than the current process time. In general, using this option only makes sense with variables recording the dates of events which occurred for all individuals before the process time begins. A typical example is date of birth with age as the corresponding duration variable.

8. As a second option, one can refer to events occurring in the currently defined sequence data structures. The syntax is

`xe =` $[k', s'_1, s'_2, d]$`,...,`

Box 4 Illustration of `seqmd` command

```
Command: seqmd(ev = [1,1,2], v=V1,V2, xe=[V1],[V2]) = out.d;
Output file:
                period
ID  Time  Z     dummies    V1   V2  V1_D V2_D
-----------------------------------------
 1    1   0    1 0 0 0 0    1 -10    0   11
 1    2   1    0 1 0 0 0    1 -10    1   12
 1    5   0    0 0 0 0 1    1 -10    4   15
 2    1   1    1 0 0 0 0    2 -15   -1   16
 2    3   0    0 0 1 0 0    2 -15    1   18
 2    4   0    0 0 0 1 0    2 -15    2   19
 2    5   1    0 0 0 0 1    2 -15    3   20
 3    3   0    0 0 1 0 0    2 -20    1   23
 3    4   1    0 0 0 1 0    2 -20    2   24
```

where k' is the number of a sequence data structure and (s'_1, s'_2) are two states specifying an event which must occur at least once in the data structure k'. Furthermore, there must be an integer, d, that will be used for the specification of effect shapes.

This option creates at least two variables. A variable $\texttt{E\_}s'_1\texttt{\_}s'_2$ that counts the number of events of type $[s'_1, s'_2]$ which occurred before the present process time; and a variable $\texttt{ED\_}s'_1\texttt{\_}s'_2$ which records the date of the last occurrence of the event $[s'_1, s'_2]$ before the current process time, or -1 if the event has not yet occurred.

If $d \geq 0$, there will be $d+1$ additional dummy variables $\texttt{E}_l\texttt{\_}s'_1\texttt{\_}s'_2$, $l = 0, \ldots, d$, defined as

$$\texttt{E}_l\texttt{\_}s'_1\texttt{\_}s'_2 = \begin{cases} 1 & \text{if } \texttt{ED\_}s'_1\texttt{\_}s'_2 \geq 0 \text{ and } t = \texttt{ED\_}s'_1\texttt{\_}s'_2 + l \\ 0 & \text{otherwise} \end{cases}$$

where t denotes the current process time; meaning that the lth dummy variable gets value 1 if the last event occurred l time units before the current process time.

To illustrate this option, we use the command

```
seqmd(ev = [1,1,2], v=V1,V2,
          xe=[V1],[V2],[1,2,1,4]) = out.d;
```

The new command file is `seq8d.cf`. The command should create additional event-specific covariates for the event $[2, 1]$ in sequence data

Box 5 Illustration of `seqmd` command

```
Command: seqmd(ev = [1,1,2], v=V1,V2, xe=[V1],[V2],[1,2,1,4])
                                                          = out.d;
Output file:
                 period
ID  Time  Z     dummies    V1   V2  V1_D V2_D E_2_1 ED_2_1  E0_2_1...
-------------------------------------------------------------------
 1    1   0   1 0 0 0 0    1 -10    0   11    0     -1     0 0 0 0 0
 1    2   1   0 1 0 0 0    1 -10    1   12    0     -1     0 0 0 0 0
 1    5   0   0 0 0 0 1    1 -10    4   15    1      4     0 1 0 0 0
 2    1   1   1 0 0 0 0    2 -15   -1   16    0     -1     0 0 0 0 0
 2    3   0   0 0 1 0 0    2 -15    1   18    1      2     0 1 0 0 0
 2    4   0   0 0 0 1 0    2 -15    2   19    1      2     0 0 1 0 0
 2    5   1   0 0 0 0 1    2 -15    3   20    1      2     0 0 0 1 0
 3    3   0   0 0 1 0 0    2 -20    1   23    1      2     0 1 0 0 0
 3    4   1   0 0 0 1 0    2 -20    2   24    1      2     0 0 1 0 0
```

structure 1; and since $d = 4$, there should be 5 dummy variables for the effect shape. Box 5 shows the resulting output file.

9. The `dtda` parameter can be used to request an additional output file containing an `nvar` command which can be used to read the data file. For example, adding the parameter `dtda=t` to the `seqmd` command used to create the data file in Box 5, would create the TDA description file shown in Box 6.

Box 6 TDA description file for data file in Box 5

```
nvar(
  dfile = out.d,
  noc = 9,
  ID   <5>[6.0] = c1 , # ID
  TIME <5>[4.0] = c2 , # time
  Z    <1>[1.0] = c3 , # target event
  P1   <1>[1.0] = c4 , # period 1
  P2   <1>[1.0] = c5 , # period 2
  P3   <1>[1.0] = c6 , # period 3
  P4   <1>[1.0] = c7 , # period 4
  P5   <1>[1.0] = c8 , # period 5
  V1 <4>[0.0] = c9,
  V2 <4>[0.0] = c10,
  V1_D <5>[4.0] = c11,
  V2_D <5>[4.0] = c12,
  E_2_1  <2>[4.0] = c13,
  ED_2_1 <2>[4.0] = c14,
  E0_2_1 <2>[4.0] = c15,
  E1_2_1 <2>[4.0] = c16,
  E2_2_1 <2>[4.0] = c17,
  E3_2_1 <2>[4.0] = c18,
  E4_2_1 <2>[4.0] = c19,
);
```

6.19 Loglinear Models

This chapter describes commands for contingency tables and loglinear models. The sections are:

6.19.1 Contingency Tables explains our notion of contingency tables and how to create such tables.

6.19.2 Loglinear Models explains the `loglin` command that can be used to estimate loglinear models.

6.19.1 Contingency Tables

Let $X_1, \ldots, X_m$ denote a set of m integer-valued variables and let

$$\tilde{X}_j = \{\tilde{x}_{1j}, \ldots, \tilde{x}_{n_j j}\}$$

denote the possible values of X_j, assumed to be in ascending order. A complete contingency tables based on these variables is defined as a table in the following way:

X_1	...	X_m	F
$\tilde{x}_{11}$	...	$\tilde{x}_{1m}$	$F(\tilde{x}_{11}, \ldots, \tilde{x}_{1m})$
$\vdots$		$\vdots$	$\vdots$
$\tilde{x}_{11}$	...	$\tilde{x}_{n_m m}$	$F(\tilde{x}_{11}, \ldots, \tilde{x}_{n_m m})$
$\vdots$		$\vdots$	$\vdots$
$\tilde{x}_{n_1 1}$	...	$\tilde{x}_{1m}$	$F(\tilde{x}_{n_1 1}, \ldots, \tilde{x}_{1m})$
$\vdots$		$\vdots$	$\vdots$
$\tilde{x}_{n_1 1}$	...	$\tilde{x}_{n_m m}$	$F(\tilde{x}_{n_1 1}, \ldots, \tilde{x}_{n_m m})$

The table has $n_1 \cdots n_m$ rows, one row for each constellation of possible values. The additional variable, F, is used to record the frequencies.

In order to create such tables one can use the `loglin` command with syntax shown in the following box. (Actually, the command has more parameters that will be explained in Section 6.19.2.)

```
loglin (
    w=...,        name of weights variable
    ptab=...,     output file for contingency table
    nfmt=...,     print format for table entries, def. 2
    maxcat=...,   max number of categories, def. 1000
) = X1,...,Xm;
```

All parameters are optional except for a list of variables on the right-hand side. It is assumed that these variables are integer-valued. The maximal

Box 1 Example data file `ll1.dat`

```
X1  X2   H
-----------
 1   2   1
 5   3   2
 2   4   3
 1   2   2
```

Box 2 Command file `ll1.cf`

```
nvar(
    dfile = ll1.dat,
    X1 = c1,
    X2 = c2,
    H  = c3,
);
loglin(
    w = H,
    ptab = ll1.tab,
) = X1,X2;
```

number of variables is 26 (corresponding to the letters A,. . . ,Z). It is assumed that each variable has at most `maxcat` different values. The command creates a complete contingency table and records its dimensions and number of non-empty cells in the standard output. As an option, the table is written to an output file specified with the `ptab` parameter. As an option, one can specify the name of a further integer-valued variable with the `w` parameter. The command then uses the values of this variable to calculate frequencies for each data matrix row.

Example 1 To illustrate, we use the data file `ll1.dat` shown in Box 1. The command file, `ll1.cf`, is shown in Box 2, and the resulting table is shown in Box 3. Note that a similar table can be creates with the `freq` command. However, the `freq` command always suppresses empty table rows. Instead, the `loglin` command always creates a complete table.

Box 3 Table created with `ll1.cf`

Idx	X1	X2	F
1	1	2	3
2	1	3	0
3	1	4	0
4	2	2	0
5	2	3	0
6	2	4	3
7	5	2	0
8	5	3	2
9	5	4	0

6.19.2 Loglinear Models

Assume a set of m Variables, $X_1, \ldots, X_m$. All variables are integer-valued, X_j has the possible values

$$\tilde{x}_{lj} \qquad l = 1, \ldots, L_j$$

As described in Section 6.19.1, a complete contingency table has m columns and

$$n = L_1 \cdots L_m$$

rows. In the following, the table will be referred to as a matrix $T = (t_{ij})$. In addition, there is a variable F such that f_i, for $i = 1, \ldots, n$, provides a count for the ith row of the table.

A loglinear model tries to explain the counts, f_i, with the help of design variables and parameters. (We follow Haberman, 1978. In fact, the algorithm behind the `loglin` command explained below is adapted from his `FREQ` program.) The modeling approach is:

$$\log(f_i) = \alpha + \sum_{k=1}^{p} d_{ik}\beta_k + \epsilon$$

where d_{ik} are values of K design variables and α and β_k are the model parameters to be estimated.

In order to explain the notion of design variables we refer to the example data in Box 1. There are two variables, X_1 and X_2, having 3 and 2 categories, respectively. The complete table has 6 cells. The variable W provides the counts. We can then set up a model like:

$$\begin{aligned}
\log(f_1) &= \alpha + d_1\beta + \epsilon \\
\log(f_2) &= \alpha + d_2\beta + \epsilon \\
\log(f_3) &= \alpha + d_3\beta + \epsilon \\
\log(f_4) &= \alpha + d_4\beta + \epsilon \\
\log(f_5) &= \alpha + d_5\beta + \epsilon \\
\log(f_6) &= \alpha + d_6\beta + \epsilon
\end{aligned}$$

containing just a single design variable. The parameters, α and β, may then be estimated with the maximum likelihood method.

Box 1 Example data file `ll2.dat`

```
X1  X2   F   S
-------------
 1   1   3   1
 1   2   4   1
 2   1   0   0
 2   2   2   1
 3   1   3   1
 3   2   5   1
```

Standard Design Vectors

As shown by the example, it is completely arbitrary how to set up a design matrix for a loglinear model. In practice, however, it is convenient to use some standardized design. The idea is to begin with a complete set of design vectors that allow for an exact fit of the counts, $f_1, \ldots, f_n$. Our approach to the definition of standardized design vectors is as follows. We treat the highest level (value) of each variable X_j as a reference category and can then define for each variable X_j a set of $L_j - 1$ design variables $D_{j1}, \ldots, D_{jL_j-1}$. Variable D_{jl} is conceived as a column vector

$$D_{jl} = \begin{pmatrix} d_{1,jl} \\ \vdots \\ d_{n,jl} \end{pmatrix}$$

with elements defined by

$$d_{i,jl} = \begin{cases} 1 & \text{if } t_{ij} = l \\ -1 & \text{if } t_{ij} = L_j \\ 0 & \text{otherwise} \end{cases}$$

Using all design vectors that can be defined in this way would result in a first-order design matrix containing the variables:

$$D = (D_{11}, \ldots, D_{1L_1}, D_{21}, \ldots, D_{2L_2-1}, \ldots, D_{m1}, \ldots, D_{mL_m-1})$$

It would be a matrix with n rows and $(L_1 - 1) \cdots (L_m - 1)$ columns. For the example data shown in Box 1 the saturated design matrix would

look as follows:

$$D = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & 1 \\ 0 & 1 & -1 \\ -1 & -1 & 1 \\ -1 & -1 & -1 \end{pmatrix}$$

In addition to first-order design vectors, one can define interaction effects. In general, second-order design vectors result from multiplying two first-order design vectors from two different variables, third-order design vectors result from multiplying three first-order design vectors from three different variables, and so on. In our example we have only two variables and can therefore only create two design vectors for interaction effects, namely $D_{11}D_{21}$ and $D_{12}D_{21}$. A complete design matrix corresponding to a saturated model for our example data would then be:

$$D_c = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & -1 & -1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & -1 & 0 & -1 \\ -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 \end{pmatrix}$$

Note that we do not take an intercept column, corresponding to α, as part of a design matrix.

Standard Approach to Model Specification

The standard approach to model specification begins with a complete design matrix that contains all first-order design vectors and, in addition, all possible interaction effects. Then subsets of this complete set of design vectors are used to define specific models. The `loglin` command that will be described below allows for this standard approach and also provides the option that the user defines an arbitrarily specified design matrix.

Notation for Model Specification

In order to allow for a flexible way to specify different models, the `loglin` command uses a specific syntax. Upper case letters,

$$A, B, C, \ldots$$

are used to refer to the variables $X_1, \ldots, X_m$ which define the contingency tables. (There can be up to 26 dimensions corresponding to the letters $A, \ldots, Z$.) A first order design vector D_{jl} is referred to by the notation $X[l]$ where X is an upper case letter corresponding to dimension j, i.e., to X_j, and l is one of the possible values of X_j, except for the highest value which is always used as a reference category. To illustrate, the design matrix D for our example could be specified by

$$A[1] + A[2] + B[1]$$

If one uses a single letter without referring to categories, this is equivalent to the design vectors for all categories (except the highest). Therefore, assuming the categories of A are $1, \ldots, L_A$, we would have

$$A \equiv A[1] + A[2] + \cdots + A[L_A - 1]$$

Both notations can be combined. In the example we might use

$$A + B[1]$$

instead of $A[1]+A[2]+B[2]$. Design variables corresponding to interaction effects can be created by simply multiplying first-order design vectors. In general:

$$X[i].Y[j].Z[k]$$

where X, Y and Z are upper case letters referring to dimensions of a table, results in a new design variable that is created from multiplying the components. For example, to create the design matrix D_c for our example, one can use:

$$A[1] + A[2] + B[1] + A[1].B[1] + A[2].B[1]$$

To simplify notation one can also use $X.Y$ to create the set of all design variables that correspond to interaction effects between the two dimensions, X and Y (any upper case letters). Therefore, a more parsimonious notation for the saturated model for our example data would be

$$A[1] + A[2] + B[1] + A.B$$

or, taking into account that X expands into all of its first-order terms, one might also write

$$A + B + A.B$$

Box 2 Syntax for `loglin` command

```
loglin (
     w=...,              name of weights variable
     ptab=...,           output file for contingency table
     nfmt=...,           print format for table entries, def. 2
     maxcat=...,         max number of categories, def. 1000

     mod=...,            model specification
     scale=...,          name of scaling variable
     mxit=...,           max number of iterations, def. 20
     tolf=...,           tolerance for convergence, def. 1.e-8
     df=...,             output file for design matrix
     pres=...,           output file for fitted table and residuals
     ppar=...,           output file for model parameters
     pcov=...,           output file for covariance matrix
     tfmt=...,           print format for parameters, def. 10.4
     mfmt=...,           print format for pcov option, def. 12.4
     screen,             screening for marginal associations
) = X1,...,Xm;
```

Finally, the notation XY might be used as an abbreviation for $X + Y + X.Y$, and the saturated model for our example data might simply be specified by AB. Note that the same syntax applies to more than two dimensions. For example, $XYZ = X(YZ)$, and so on.

The `loglin` Command

The syntax for the `loglin` command is shown in Box 2. Except for a list of variable names on the right-hand side, all other parameters are optional. If no further parameters are used, the command creates a contingency tables as explained in Section 6.19.1.

1. The `screen` parameter can be used to request a calculation of partial and marginal associations based on all hierarchical loglinear model. See Lustbader and Stodola [1981] for further details.

2. The user can request model estimation by providing a model specification with the `mod` parameter. (This parameter can be used up to 50 times in the same `loglin` command.) The syntax is:

```
mod = design_matrix,
```

There are two possibilities to specify a design matrix. One possibility is to directly use the syntax explained above. Alternatively, one can use the parameter as

```
mod = d:fname,
```

where `fname` is the name of a file that contains a design matrix. It is assumed that the file has free-format numerical entries and the number of records equals the number of rows of the contingency table. Note that the file may only contain integers. In fact, TDA stores a design matrix always in short integers (range is about ±32000).

3. If requested with the `df` parameter, the design matrix actually used for model estimation is written into an output file. If the `mod` parameter is used more than once, a separate design matrix is written for each of the models.

4. Model estimation is done by maximum likelihood as described by Haberman [1978]. The maximum number of iterations can be controlled with the `itmx` parameter, the convergence tolerance with the `tolf` parameter.

5. Estimated parameters and standard errors are written into the standard output. Parameter estimated will be written into an output file if requested with the `ppar` parameter; the covariance matrix will be written into an output file if requested with the `pcov` parameter.

6. In addition one can request a further output file with the `pres` parameter. It will contain the contingency table and furthermore the fitted counts and standardized residuals as explained in Haberman [1978, II, p. 78].

7. As a further option one can provide the name of a scaling variable with the `scale` parameter. The estimated model is then

$$\log(f_i/s_i) = \alpha + \sum_{k=1}^{p} d_{ik}\beta_k + \epsilon$$

where s_i are the values of the scaling variable. If $s_i \leq 0$, the fitted value, $\hat{f}_i$, will be zero. So this option can also be used for tables that contain "'structural zeroes"'.

Example 1 To illustrate, we fit a main effects loglinear model to the data shown in Box 1. The command file, `ll2.cf`, is shown in Box 3. Part

Box 3 Command file `ll2.cf`

```
nvar(
    dfile = ll2.dat,
    X1 = c1,
    X2 = c2,
    F  = c3,
);
loglin(
    w = F,
    mod = A+B,
    pres = res,
) = X1,X2;
```

Box 4 Fitted values and residuals.

```
Residuals of Model: A+B

Index   A  B   Observed       Fitted     Residual        Scale
-----------------------------------------------------------------
    1   1  1          3       2.4706       0.5459       1.0000
    2   1  2          4       4.5294      -0.5459       1.0000
    3   2  1          0       0.7059      -1.1119       1.0000
    4   2  2          2       1.2941       1.1119       1.0000
    5   3  1          3       2.8235       0.1794       1.0000
    6   3  2          5       5.1765      -0.1794       1.0000
```

of the standard output is shown in Box 5 and the file requested with the `pres` option is shown in Box 4.

Box 5 Part of standard output (command file `ll2.cf`)

```
Loglinear models.
Definition and structure of contingency table.
Frequencies defined by: F

Dimension  Variable   Categories
  1  A     X1           3 :  1  2  3
  2  B     X2           2 :  1  2

Table has 6 cells, 17 counts.
Table is incomplete: 5 (83.33%) of 6 cells.

Check of model requests.
Begin of model estimation.

Model: A+B

Number of model parameters: 4
Maximum number of iterations: 20
Tolerance for convergence: 1e-08

Convergence reached in 7 iterations

Likelihood Ratio Statistic         1.9287  Prob: 0.3812
Pearson's Chi Square               1.2833  Prob: 0.5264
F-Statistic                        0.9644
Degrees of Freedom 2

Idx  Parameter      Coeff      Error     T-Stat  Signif
------------------------------------------------------
  0  Constant      0.8344        ---        ---     ---
  1  A[1]          0.3731     0.3646     1.0232  0.6938
  2  A[2]         -0.8797     0.5020    -1.7524  0.9203
  3  B[1]         -0.3031     0.2538    -1.1943  0.7676

Residuals written to: res
```

7. Relational Data

This part contains the following sections.

7.1 Introduction and Overview

Command		Section
gcd	creating simple test data	3.6.3.1
gcliq	standard cliques	7.2.9.1
gcon	connected components	7.2.3.1
gcset	compact sets	7.2.9.2
gcut	cut nodes and blocks	7.2.3.3
gcyc	fundamental set of cycles	7.2.6.1
gdcon	reachable nodes in digraphs	7.2.3.2
gdcyc	enumeration of cycles	7.2.6.2
gdd	definition of data structure	3.6.2
gdln	direct links	7.2.1.2
gdp	writing relational data	3.6.4.1
gep	enumeration of paths	7.2.4.1
gev	eigenvalues and eigenvectors	7.2.7
gflow	maximal flows	7.2.8.1
giset	independent sets	7.2.9.3
gmst	minimum spanning tree	7.2.5.2
gnst	enumeration of spanning tree	7.2.5.3
gni	degree of nodes	7.2.1.1
gsort	topological sort	7.2.2.1
gsp	all shortest paths	7.2.4.2
gst	depth-first spanning trees	7.2.5.1
gtcl	transitive closure	7.2.4.3
gap	column permutations	7.3.1.1
gqap	quadratic assignment	7.3.1.2
becl	bond energy clustering	7.5.3.1
hcld	simple divisive clustering	7.5.2.1
hcld	partition with minimal diameter	7.5.2.2
hcls	SAHN clustering procedures	7.5.1.1
nncl	nearest-neighbor clustering	7.5.1.2
gbcf	direct/indirect backward control	7.6.1.1
gfc	measures of flow control	7.6.1.3
gfcf	direct/indirect forward control	7.6.1.1
gio	integrated ownership	7.6.1.2

7.2 General Graph Algorithms

This section deals with general procedures for describing and analyzing graphs. Available commands will be described in the following subsections.

7.2.1 Direct Links describes commands to find direct forward and backwards links of the nodes of a graph, and to calculate the degree of nodes.

7.2.2 Partial Orders deals with specific questions concerning partially ordered sets, e.g., topological sorting of a directed graph.

7.2.3 Connectivity describes commands to investigate connectivity of a graph and to find its components, cut nodes, and blocks.

7.2.4 Paths describes commands to find paths and, in particular, all shortest paths connecting the nodes of a graph.

7.2.5 Spanning Trees describes commands to find spanning trees and, in particular, minimum spanning trees in undirected graphs.

7.2.6 Cycles describes commands that find all cycles in directed and undirected graphs.

7.2.7 Eigenvalues and Eigenvectors describes a command that calculates a subset of the eigenvalues and eigenvectors of a graph's adjacency matrix.

7.2.8 Flows describes commands that can be used to calculate maximal flows in networks.

7.2.9 Subgroups describes commands to find subgroups of a graph. This includes cliques, compact sets, and independent sets.

7.2.1 Direct Links

This section describes two simple commands that can be used to get information about the nodes of a graph.

7.2.1.1 Degree of Nodes describes the `gni` command that calculates the degree of the nodes in a graph.

7.2.1.2 Direct Links describes the `gdln` command that can be used to find the nodes that are directly connected with a given set of nodes.

Further commands, specifically designed to characterize nodes of a graph, will be described in the section on social network analysis.

7.2.1.1 Degree of Nodes

Given a graph, or multigraph, defined with the `gdd` command, one can use the `gni` command to find for each node its degree. If the graph is directed, the command calculates in-degrees and out-degrees separately. The syntax is shown in the following box.

```
gni (
    gn=...,          graph number
    nfmt=...,        integer print format, def. 4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. By default, the command calculates degrees for all graphs contained in the currently defined multigraph. Alternatively, one can specify a graph number with the `gn` parameter.

The first two columns of the output file contain, respectively, the internal and external node numbers. Then follow, for each graph, three further columns.

1. The first column shows the in-degree of the node,
2. the second column shows the out-degree of the node, and
3. the third column shows whether the node has a loop.

Note that loops are not counted when calculating in-degrees and out-degrees of a node. If the graph is undirected, in-degrees and out-degrees will be identical.

Example 1 To illustrate we use the multigraph data file `gd1.dat`, see Box **3.6.2**-2. The command file is `gd4.cf`. Box 1 shows the resulting output files, based on interpreting the graphs as directed and undirected, respectively.

Box 1 Output files created by `gni` command (example 1)

graph is directed

i	N(i)	ID1	OD1	NL1	ID2	OD2	NL2
1	1	1	2	0	2	0	0
2	5	1	0	0	0	1	0
3	7	1	1	0	1	1	0
4	8	1	1	1	0	1	0
5	9	0	0	0	0	0	0
6	11	0	1	0	1	0	0
7	12	1	0	0	0	1	0

graph is undirected

i	N(i)	ID1	OD1	NL1	ID2	OD2	NL2
1	1	2	2	0	2	2	0
2	5	1	1	0	1	1	0
3	7	1	1	0	2	2	0
4	8	0	0	1	1	1	0
5	9	0	0	0	0	0	0
6	11	1	1	0	1	1	0
7	12	1	1	0	1	1	0

7.2.1.2 Direct Links

When working with large directed graphs, one sometimes needs information about the set of nodes that can be directly reached, or are reachable from, other nodes. The `gdln` command provides this information. The command requires a directed graph, defined as an edge list (option 1 in the `gdd` command). The syntax is shown in the following box.

```
gdln (
    opt=...,        option, def. 1
                    1 = forward links
                    2 = backward links
    nfmt=...,       integer print format, def. 4
    gn=...,         graph number, def. 1
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. By default, the command uses the first graph (`gn=1`) in the currently defined multigraph. There are two options.

Option 1. creates information about forward links. The output file will contain one record for each node that is the starting point for at least one edge (including loops). The first two columns contain respectively, the internal and external node number. The third column contains the number of edges, say n_i, that begin in the current node. Then follow n_i columns containing the numbers of those nodes that can be directly reached from the current node.

Option 2. creates information about backward links. The output file will contain one record for each node that can be reached by at least one other node (including loops). The first two columns contain respectively, the internal and external node number. The third column contains the number of edges, say n_i, that end in the current node. Then follow n_i columns containing the numbers of those nodes that can directly reach the current node.

Box 1 Output files created with `gdln` command.

Forward links

i	N(i)	NF	L1	L2
1	1	2	5	7
3	7	1	1	
4	8	1	8	
6	11	1	12	

Backward links

i	N(i)	NB	L1
1	1	1	7
2	5	1	1
3	7	1	1
4	8	1	8
7	12	1	11

Example 1 To illustrate we use the multigraph data file `gd1.dat`, see Box **3.6.2**-2. The command file is `gd5.cf`. Box 1 shows the resulting output files.

7.2.2 Partial Orders

This section is intended to discuss approaches to partially ordered sets. Currently, there is only a single subsection.

7.2.2.1 Topological Sorting describes a command that can be used for a topological sorting of a directed graph.

7.2.2.1 Topological Sorting

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ denote a directed graph. For each $i \in \mathcal{N}$, define

$$R(i) = \{j \in \mathcal{N} \mid j \neq i,\ \exists\,(i,j) \in \mathcal{E}\}$$

If the graph $\mathcal{G}$ has no cycles it is possible to find new labels for its nodes, say $l(i)$ for $i \in \mathcal{N}$, in such a way that

$$\forall\, i \in \mathcal{N}\ :\ j \in R(i) \implies l(j) > l(i)$$

Using the new labels for representing the graph is then called *topological sorting* of $\mathcal{G}$. The `gsort` command uses an algorithm described in Neumann and Morlock [1993, p. 198]. The syntax is shown in the following box.

```
gsort (
    opt=...,        option, def. 1
                    1 = list of old/new labels
                    2 = graph with new labels
    gn=...,         graph number, def. 1
    nfmt=...,       integer print format, def. 4
    fmt=...,        print format for values, def. 10.4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify a directed graph that is defined by option 1 of the `gdd` command, i.e., by an edge list. If the algorithm finds a cycle it stops with an error message. So the command also provides an easy way to check whether a directed graph contains cycles.

Example 1 To illustrate we use the graph shown in Figure 1. The data file `gd6.dat` (Box 1) is used to define a `gdd` data structure with option 1 (edge list). The command file is `gd22.cf`. Box 2 shows the output files for the two options. $N(i)$ and $L(i)$ denote the old and new labels, respectively.

Box 1 Data file `gd6.dat`

```
 I  J  V
---------
 5  7  1
 4  7  5
 4  5  6
 5  1  3
 7  1  2
 1  2  8
 7  2  4
 4  2  7
```

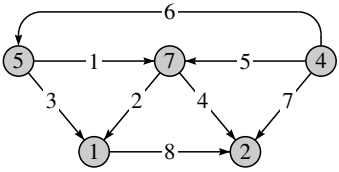


Figure 1 Graph (plotted with `gd21.cf`) of the data shown in Box 1.

Box 2 Output files from `gsort` command

```
    option 1                    option 2

 i   N(i) L(i)      L(i) L(j) N(i) N(j)   value
--------------      ----------------------------
 1    1    4         1    2    4    5     6.0000
 2    2    5         1    3    4    7     5.0000
 3    4    1         1    5    4    2     7.0000
 4    5    2         2    3    5    7     1.0000
 5    7    3         2    4    5    1     3.0000
                     3    4    7    1     2.0000
                     3    5    7    2     4.0000
                     4    5    1    2     8.0000
```

7.2.3 Connectivity

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ denote an undirected graph. A *walk* in $\mathcal{G}$ is a sequence of nodes, $(i_1, \ldots, i_m)$, all contained in $\mathcal{N}$ and each two consecutive nodes are connected by an edge in $\mathcal{E}$. A walk is called a *path* (from i_1 to i_m) if all nodes in the sequence are different. A walk is called a *cycle*, if all nodes in the sequence are different and $i_1 = i_m$.

Two nodes $i, j \in \mathcal{N}$ are said to be *connected* if there exists a path from i to j. In an undirected graph, if there is a path from i to j, then there is also a path from j to i.

A graph is called *connected* if every pair of its nodes is connected, otherwise it is called *disconnected*. As is easily seen, in an undirected graph the relation "connected to" defines an equivalence relation on the set of nodes, $\mathcal{N}$; and this in turn defines a partition of $\mathcal{N}$ into equivalence classes. The subgraphs induced by these equivalence classes are called *components* of the graph. Alternatively, one can say that a component is a maximally connected subgraph. Of course, a connected graph has only one component, identical with itself.

Let now $\mathcal{G}$ be a directed graph. For every two nodes $i, j \in \mathcal{N}$ one can then distinguish a path from i to j, and a path from j to i. Two nodes, i and j, are said to be *strongly connected* if there is both a path from i to j and from j to i. Again, "strongly connected with" is an equivalence relation on $\mathcal{N}$; the subgraphs induced by its equivalence classes are called the *strongly connected components* of the directed graph $\mathcal{G}$.

Two nodes are said to be *weekly connected* if they are connected in the corresponding underlying undirected graph. The connected components in this underlying graph are called the *weekly connected components* of the directed graph.

This section describes commands that can be used to investigate connectivity in both undirected and directed graphs.

7.2.3.1 Connected Components describes the `gcon` command that finds the connected components in an undirected graph.

7.2.3.2 Reachable Nodes in Digraphs describes the `gdcon` command that finds, in directed graphs, all nodes that are reachable from a given set of nodes.

7.2.4.3 Cut Nodes and Blocks describes the `gcut` command that finds the cut nodes and blocks in an undirected graph.

7.2.3.1 Connected Components

This section describes the `gcon` command that can be used to find the connected components in an undirected graph. The algorithm is based on a version of depth-first search; for a general introduction see, e.g., Sedgewick [1990, ch. 29]. The graph can be defined with any of the options provided by the `gdd` command. The syntax is shown in the following box.

```
gcon(
    opt=...,        option, def. 1
                    1 = node list
                    2 = edge list
    gn=...,         graph number, def. 1
    nfmt=...,       integer print format, def. 4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. By default, the command uses the first graph (`gn=1`) in the currently defined multigraph. The contents of the output file depends on the `opt` parameter. By default (`opt=1`), the command creates a node list. The first column of the output file numbers the connected components, then follows the number of nodes in the component, and finally the internal and external node numbers. If `opt=2`, the data for each component are written as an edge list.

Example 1 To illustrate the `gcon` command we use the graph shown in Box **3.6.2**-2 and Figure **3.6.2**-2. The command file, `gd7.cf`, first uses these data to define an undirected graph, and then uses the `gcon` command to request information about the connected components in the second graph. The output files for both options 1 and 2 are is shown in Box 1. There are three components. The first one contains the nodes 1, 5, 7, and 8. The second and third components contain one and two nodes, respectively.

Box 1 Output files created by `gcon` command

option 1

C	N	i	N(i)
1	4	1	1
1	4	2	5
1	4	3	7
1	4	4	8
2	1	5	9
3	2	6	11
3	2	7	12

option 2

C	N	i	j	N(i)	N(j)	value
1	4	1	2	1	5	4.0000
1	4	1	3	1	7	15.0000
1	4	3	4	7	8	0.0000
2	1	5	5	9	9	-1.0000
3	2	6	7	11	12	14.0000

7.2.3.2 Reachable Nodes in Digraphs

This section describes the `gdcon` command that finds, for each node i, the set of other nodes that can be reached by a directed path from i. The graph must be directed, but can be defined with any of the options provided by the `gdd` command. The algorithm is based on a version of depth-first search. The syntax is shown in the following box.

```
gdcon (
    opt=...,        option, def. 1
                    1 = number of reachable nodes
                    2 = plus list of node numbers
    gn=...,         graph number, def. 1
    nfmt=...,       integer print format, def. 4
    if=...,         input file with nodes numbers
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. By default, the command uses the first graph (`gn=1`) in the currently defined multigraph. If the `if` parameter is not used, the command performs its calculations for all nodes in the input graph. Alternatively, one can specify the name of an input file with the `if` parameter. The first numerical entry in each of its records is then interpreted as a node number, and the command only considers these node numbers.

The output file will contain one record for each node number that has an out-degree of at least 1. The first two columns show, respectively, the internal and external node number. The third column contains the number of nodes which are reachable by a directed path. If `opt=2`, further columns show the numbers of the nodes that are reachable.

Example 1 To illustrate the `gdcon` command we use the graph shown in Box 3.6.2-2 and Figure 3.6.2-2. The command file, `gd8.cf`, first uses these data to define a directed graph, and then uses the command

```
gdcon (gn=2,opt=2) = d;
```

Box 1 Output file created by `gdcon` command

```
 i  N(i)   N    node numbers
---------------------------
 2    5    1     1
 3    7    1     1
 4    8    2     1    7
 7   12    1    11
```

The output file, `d`, is shown in Box 1.

7.2.3.3 Cut Nodes and Blocks

Consider the connected components of an undirected graph. A node is called a *cut node*, or *articulation point*, if removing the node makes the component disconnected. If a connected component does not have cut points it is called a *block*. For a discussion of these notions see, e.g., Gibbons [1985, ch. 1].

To find the cut nodes and blocks in an undirected graph one can use the `gcut` command with syntax shown in the following box. The algorithm is a version of depth-first search as described in Sedgewick [1990, pp. 423–427].

```
gcut (
    opt=...,        option, def. 1
                    1 = one record for each component
                    2 = one record for each node
    gn=...,         graph number, def. 1
    nfmt=...,       integer print format, def. 4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify an undirected graph. The data written to the output file depends on the `opt` parameter and will be explained in the example below.

Example 1 To illustrate we use the graph shown in Figure 7.2.5.1-1. Since the graph doesn't contain cut nodes we remove edges $(3, 4)$ and $(3, 5)$. The new data file is `gdat8.dat`. The command file is `gd26.cf`. Box 1 shows the output files from the `gcut` command for both options.

If `opt=1`, the output file will contain one record for each component. The first column, labeled `C`, counts the components. The second column, labeled `N`, shows the number of elements in the component. The third column, labeled `CN`, shows the number of cut nodes in the component. Then follow the external node numbers of the cut nodes. In this example, the first component has two cut nodes, the second component is a block.

If `opt=2`, the output file contains one record for each node in the

Box 1 Output files from `gcut` command

option 1

C	N	CN	cut nodes	
1	5	2	2	4
2	3	0		

option 2

C	i	N(i)	CN	F
1	1	1	2	0
1	2	2	2	1
1	3	3	2	0
1	4	4	2	1
1	5	5	2	0
2	6	6	0	0
2	7	7	0	0
2	8	8	0	0

graph. The first three columns give the index number of the component and the internal and external node number. `CN` is the number of cut nodes in the component, and `F` indicates whether the current node is a cut node.

7.2.4 Paths

This section describes commands that can be used to investigate paths in directed and undirected graphs.

7.2.4.1 Enumeration of Paths describes a command that finds all paths in a graph.

7.2.4.2 Shortest Paths describes a command that finds all shortest paths in a graph.

7.2.4.3 Transitive Closure describes a command that finds the transitive closure of a graph.

7.2.4.1 Enumeration of Paths

The `gep` command can be used to find all paths connecting any two nodes of a graph and to determine their minimal (or maximal) length and/or value. The graph can be directed or undirected but must be defined with `gdd` option 1 (edge list). The algorithm is based on a version of depth-first search and ignores loops. The syntax is shown in the following box.

```
gep (
   opt=...,      output option, def. 1
                 1 = one record for each pair of nodes
                 2 = all paths
                 3 = distance matrix, minimal values
                 4 = distance matrix, maximal values
                 5 = location of nodes in paths, version 1
                 6 = location of nodes in paths, version 2
   max=...,      max number of paths (opt 6), def. 100
   gn=...,       graph number, def. 1
   nfmt=...,     integer print format, def. 4
   fmt=...,      print format for values, def. 10.4
   if=...,       input file for node selection
   ns=...,       max edges in input file, def. 100
) = fname;
```

Except for the name of an output file on the right-hand side all parameters are optional. By default, the command considers all pairs of nodes in the given graph. Optionally, one can specify a subset of nodes with the `if` parameter. If a file is specified with this parameter, the command tries to interpret the first two integer entries in each of its records as external node numbers i and j, respectively, and then only considers paths from i to j. If this option is used, the command needs to know a maximal number of records (pairs of nodes) from the input file. This can be specified with the `ns` parameter; default is a maximum of 100 pairs of nodes.

In order to illustrate the options we shall use the graph shown in Figure 1. (The data file is `gd5.dat`, the plot was creates with command

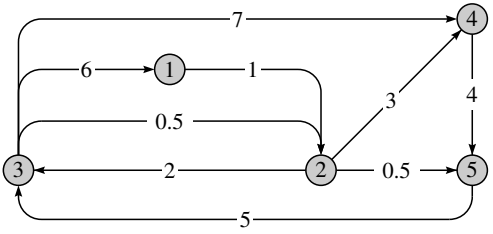


Figure 1 A directed graph used for examples (`gd18.cf`).

file `gd18.cf`.) The command file for the examples is `gd20.cf`.

1. If `opt = 1` (default), the output file will contain one record for each pair of nodes that is connected by at least one path. In this case, the columns are as follows:

1. Internal node number of first node, i.
2. Internal node number of second node, j.
3. External node number of first node.
4. External node number of second node.
5. Number of paths connecting i and j.
6. Number of paths connecting i and j that have minimal value.
7. Minimal length of a path connecting i and j.
8. Minimal value of a path connecting i and j.
9. Maximal length of a path connecting i and j.
10. Maximal value of a path connecting i and j.

This option is illustrated by the output file shown in Box 1. For instance, there are three paths connecting nodes 1 and 3. The shortest path has value 3 and contains two edges.

2. If `opt = 2`, the output file will contain one record for each path that connects two nodes of the graph. In this case, the columns are as follows:

1. Internal node number of first node, i.
2. Internal node number of second node, j.
3. External node number of first node.
4. External node number of second node.
5. Number of the path connecting i and j.

Box 1 Output file from `gep` command, option 1

i	j	N(i)	N(j)	NP	NPS	MINLEN	MINVAL	MAXLEN	MAXVAL
1	2	1	2	1	1	1	1.0000	1	1.0000
1	3	1	3	3	1	2	3.0000	4	13.0000
1	4	1	4	3	1	2	4.0000	4	13.5000
1	5	1	5	3	1	2	1.5000	4	14.0000
2	1	2	1	3	1	2	8.0000	4	18.0000
2	3	2	3	3	1	1	2.0000	3	12.0000
2	4	2	4	3	1	1	3.0000	3	12.5000
2	5	2	5	3	1	1	0.5000	3	13.0000
3	1	3	1	1	1	1	6.0000	1	6.0000
3	2	3	2	2	1	1	0.5000	2	7.0000
3	4	3	4	3	1	2	3.5000	3	10.0000
3	5	3	5	5	1	2	1.0000	4	14.0000
4	1	4	1	1	1	3	15.0000	3	15.0000
4	2	4	2	2	1	3	9.5000	4	16.0000
4	3	4	3	1	1	2	9.0000	2	9.0000
4	5	4	5	1	1	1	4.0000	1	4.0000
5	1	5	1	1	1	2	11.0000	2	11.0000
5	2	5	2	2	1	2	5.5000	3	12.0000
5	3	5	3	1	1	1	5.0000	1	5.0000
5	4	5	4	3	1	3	8.5000	4	15.0000

6. Length of the path.
7. Value of the path.
8. Sequence of external node numbers contained in the path.

This option is illustrated in Box 2. For example, there is now a separate record for each of the three paths connecting nodes 1 and 3.

3. If `opt = 3`, the output file will contain one record for each node of the graph. The first two columns contain, respectively, the internal and external node number. Then follow n (= number of nodes) columns. Being in record i, column $j+2$ contains the minimal value of the paths connecting i and j; or -1 if there is no path connecting i and j. If $i = j$, the entry will always be zero. This option is illustrated in the upper half of Box 3.

4. If `opt = 4`, the output file will the same format as with option 3. The only difference is that the entries will show the maximal distance, instead of the minimal distance. For an illustration, see the lower part of Box 3.

Box 2 Output file from `gep` command, option 2

```
i    j  N(i) N(j)  NP  MINLEN   MINVAL    nodes in path
-------------------------------------------------------------
1    2    1    2    1    1      1.0000    1    2
1    3    1    3    1    2      3.0000    1    2    3
1    3    1    3    2    4     13.0000    1    2    4    5    3
1    3    1    3    3    3      6.5000    1    2    5    3
1    4    1    4    1    3     10.0000    1    2    3    4
1    4    1    4    2    2      4.0000    1    2    4
1    4    1    4    3    4     13.5000    1    2    5    3    4
1    5    1    5    1    4     14.0000    1    2    3    4    5
1    5    1    5    2    3      8.0000    1    2    4    5
1    5    1    5    3    2      1.5000    1    2    5
2    1    2    1    1    2      8.0000    2    3    1
2    1    2    1    2    4     18.0000    2    4    5    3    1
2    1    2    1    3    3     11.5000    2    5    3    1
2    3    2    3    1    1      2.0000    2    3
2    3    2    3    2    3     12.0000    2    4    5    3
2    3    2    3    3    2      5.5000    2    5    3
2    4    2    4    1    2      9.0000    2    3    4
2    4    2    4    2    1      3.0000    2    4
2    4    2    4    3    3     12.5000    2    5    3    4
2    5    2    5    1    3     13.0000    2    3    4    5
2    5    2    5    2    2      7.0000    2    4    5
2    5    2    5    3    1      0.5000    2    5
3    1    3    1    1    1      6.0000    3    1
3    2    3    2    1    2      7.0000    3    1    2
3    2    3    2    2    1      0.5000    3    2
3    4    3    4    1    3     10.0000    3    1    2    4
3    4    3    4    2    2      3.5000    3    2    4
3    4    3    4    3    1      7.0000    3    4
3    5    3    5    1    4     14.0000    3    1    2    4    5
3    5    3    5    2    3      7.5000    3    1    2    5
3    5    3    5    3    3      7.5000    3    2    4    5
3    5    3    5    4    2      1.0000    3    2    5
3    5    3    5    5    2     11.0000    3    4    5
4    1    4    1    1    3     15.0000    4    5    3    1
4    2    4    2    1    4     16.0000    4    5    3    1    2
4    2    4    2    2    3      9.5000    4    5    3    2
4    3    4    3    1    2      9.0000    4    5    3
4    5    4    5    1    1      4.0000    4    5
5    1    5    1    1    2     11.0000    5    3    1
5    2    5    2    1    3     12.0000    5    3    1    2
5    2    5    2    2    2      5.5000    5    3    2
5    3    5    3    1    1      5.0000    5    3
5    4    5    4    1    4     15.0000    5    3    1    2    4
5    4    5    4    2    3      8.5000    5    3    2    4
5    4    5    4    3    2     12.0000    5    3    4
```

Box 3 Output files from `gep` command, options 3 and 4

```
i  N(i)    option 3
-------------------------------------------------------------
1    1      0.0000      1.0000      3.0000      4.0000      1.5000
2    2      8.0000      0.0000      2.0000      3.0000      0.5000
3    3      6.0000      0.5000      0.0000      3.5000      1.0000
4    4     15.0000      9.5000      9.0000      0.0000      4.0000
5    5     11.0000      5.5000      5.0000      8.5000      0.0000

i  N(i)    option 4
-------------------------------------------------------------
1    1      0.0000      1.0000     13.0000     13.5000     14.0000
2    2     18.0000      0.0000     12.0000     12.5000     13.0000
3    3      6.0000      7.0000      0.0000     10.0000     14.0000
4    4     15.0000     16.0000      9.0000      0.0000      4.0000
5    5     11.0000     12.0000      5.0000     15.0000      0.0000
```

5. In social network analysis, one is sometimes interested in the characterization of nodes by their ability to "control" communication between other nodes.[1] One version of this idea simply investigates the location of nodes in paths connecting other nodes. This version is supported by options 5 and 6 of the `gep` command. Option 5 considers all shortest paths, option 6 takes all paths into account.

If `opt = 5`, the output file will contain one record for each pair of nodes that is connected by at least one paths. The columns are:

1. Internal node number of first node, i.
2. Internal node number of second node, j.
3. External node number of first node.
4. External node number of second node.
5. Number of paths connecting i and j.
6. Number of paths connecting i and j that have minimal value.
7. Minimal length of a path connecting i and j.
8. Minimal value of a path connecting i and j.
9. Entries n_k, for $k = 1, \ldots, n$; n being the number of nodes in the graph, and the nodes are sorted in ascending order. n_k is the number of times node k occurs in a path of minimal length connecting nodes i and j, i.e., the pair of nodes referred to in the current record.

An illustration is given in Box 4, based on the graph shown in Figure 1, interpreted as directed and valued.

[1] See, e.g., the discussion in Freeman [1978].

Box 4 Output from `gep` command, option 5

i	j	N(i)	N(j)	NP	NPS	MINLEN	MINVAL	1	2	3	4	5
1	2	1	2	1	1	1	1.00	0	0	0	0	0
1	3	1	3	3	1	2	3.00	0	1	0	0	0
1	4	1	4	3	1	2	4.00	0	1	0	0	0
1	5	1	5	3	1	2	1.50	0	1	0	0	0
2	1	2	1	3	1	2	8.00	0	0	1	0	0
2	3	2	3	3	1	1	2.00	0	0	0	0	0
2	4	2	4	3	1	1	3.00	0	0	0	0	0
2	5	2	5	3	1	1	0.50	0	0	0	0	0
3	1	3	1	1	1	1	6.00	0	0	0	0	0
3	2	3	2	2	1	1	0.50	0	0	0	0	0
3	4	3	4	3	1	2	3.50	0	1	0	0	0
3	5	3	5	5	1	2	1.00	0	1	0	0	0
4	1	4	1	1	1	3	15.00	0	0	1	0	1
4	2	4	2	2	1	3	9.50	0	0	1	0	1
4	3	4	3	1	1	2	9.00	0	0	0	0	1
4	5	4	5	1	1	1	4.00	0	0	0	0	0
5	1	5	1	1	1	2	11.00	0	0	1	0	0
5	2	5	2	2	1	2	5.50	0	0	1	0	0
5	3	5	3	1	1	1	5.00	0	0	0	0	0
5	4	5	4	3	1	3	8.50	0	1	1	0	0

6. If `opt = 6`, similar information will be given for all path connecting each pair of nodes in the graph. The output file will contain one record for each path in the graph. The columns are:

1. Internal node number of first node, i.
2. Internal node number of second node, j.
3. External node number of first node.
4. External node number of second node.
5. Total number of paths connecting i and j.
6. Current number of path connecting i and j.
7. Length of the path.
8. Value of the path.
9. Entries n_k, for $k = 1, \ldots, n$; n being the number of nodes in the graph, and the nodes are sorted in ascending order. $n_k = 1$ if node k occurs in the path connecting nodes i and j, i.e., the pair of nodes referred to in the current record.

For each pair of nodes, i and j, the paths are sorted in ascending order according to their value. An illustration is given in Box 5, based on the graph shown in Figure 1, interpreted as directed and valued.

Box 5 Output from gep command, option 6

i	j	N(i)	N(j)	NP	MINLEN	MINVAL	1	2	3	4	5
1	2	1	2	1	1	1.00	0	0	0	0	0
1	3	1	3	3	1	2	3.00	0	1	0	0	0
1	3	1	3	3	2	3	6.50	0	1	0	0	1
1	3	1	3	3	3	4	13.00	0	1	0	1	1
1	4	1	4	3	1	2	4.00	0	1	0	0	0
1	4	1	4	3	2	3	10.00	0	1	1	0	0
1	4	1	4	3	3	4	13.50	0	1	1	0	1
1	5	1	5	3	1	2	1.50	0	1	0	0	0
1	5	1	5	3	2	3	8.00	0	1	0	1	0
1	5	1	5	3	3	4	14.00	0	1	1	1	0
2	1	2	1	3	1	2	8.00	0	0	1	0	0
2	1	2	1	3	2	3	11.50	0	0	1	0	1
2	1	2	1	3	3	4	18.00	0	0	1	1	1
2	3	2	3	3	1	1	2.00	0	0	0	0	0
2	3	2	3	3	2	2	5.50	0	0	0	0	1
2	3	2	3	3	3	3	12.00	0	0	0	1	1
2	4	2	4	3	1	1	3.00	0	0	0	0	0
2	4	2	4	3	2	2	9.00	0	0	1	0	0
2	4	2	4	3	3	3	12.50	0	0	1	0	1
2	5	2	5	3	1	1	0.50	0	0	0	0	0
2	5	2	5	3	2	2	7.00	0	0	0	1	0
2	5	2	5	3	3	3	13.00	0	0	1	1	0
3	1	3	1	1	1	1	6.00	0	0	0	0	0
3	2	3	2	2	1	1	0.50	0	0	0	0	0
3	2	3	2	2	2	2	7.00	1	0	0	0	0
3	4	3	4	3	1	2	3.50	0	1	0	0	0
3	4	3	4	3	2	1	7.00	0	0	0	0	0
3	4	3	4	3	3	3	10.00	1	1	0	0	0
3	5	3	5	5	1	2	1.00	0	1	0	0	0
3	5	3	5	5	2	3	7.50	1	1	0	0	0
3	5	3	5	5	3	3	7.50	0	1	0	1	0
3	5	3	5	5	4	2	11.00	0	0	0	1	0
3	5	3	5	5	5	4	14.00	1	1	0	1	0
4	1	4	1	1	1	3	15.00	0	0	1	0	1
4	2	4	2	2	1	3	9.50	0	0	1	0	1
4	2	4	2	2	2	4	16.00	1	0	1	0	1
4	3	4	3	1	1	2	9.00	0	0	0	0	1
4	5	4	5	1	1	1	4.00	0	0	0	0	0
5	1	5	1	1	1	2	11.00	0	0	1	0	0
5	2	5	2	2	1	2	5.50	0	0	1	0	0
5	2	5	2	2	2	3	12.00	1	0	1	0	0
5	3	5	3	1	1	1	5.00	0	0	0	0	0
5	4	5	4	3	1	3	8.50	0	1	1	0	0
5	4	5	4	3	2	2	12.00	0	0	1	0	0
5	4	5	4	3	3	4	15.00	1	1	1	0	0

Contrary to the other options, if `opt = 6`, the command needs to know an upper limit for the number of paths connecting any pair of nodes. This maximal number of paths can be specified with the `max` parameter, default is `max = 100`. If for any pair of nodes, the command finds more than this number of paths, it exits with an error message.

7.2.4.2 All Shortest Paths

The section describes the `gsp` command that finds, for each pair of nodes in a graph, a shortest path. The graph can be directed or undirected, valued or unvalued. If the graph is unvalued, all edge values are assumed to be 1. The algorithm is adapted from Pape [1980]. The syntax of the command is shown in the following box.

```
gsp (
    opt=...,        option, def. 1
                    1 = only reachable nodes
                    2 = square matrix with leading columns
                    3 = square matrix without leading columns
    gn=...,         graph number, def. 1
    nfmt=...,       integer print format, def. 4
    fmt=...,        print format for values, def. 10.4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. In order to illustrate the options we use the example data from Section 7.2.4.1.

1. If `opt=1` (default), the output file will contain one record for each node of the graph. The first two columns show, respectively, the internal and external node numbers. Then follows, for each node that can be reached from the current node, its external node number and its shortest distance from the current node. Box 1 provides an illustration.

2. If `opt=2`, the output file will contain an (n, n) square matrix with entries showing the shortest distances between each pair of nodes. n is the number of nodes in the graph. In addition, two leading columns show the internal and external node numbers corresponding to each row of the matrix.

3. The third option is identical to `opt=2` except that the leading two columns are not written to the output file. This allows to read the output file directly into a square distance matrix.

Box 1 Output file from `gsp` command, option 1

```
i  N(i)   reachable nodes and shortest distances
-------------------------------------------------------
1    1    2   1.0     3   3.0     4   4.0     5   1.5
2    2    1   8.0     3   2.0     4   3.0     5   0.5
3    3    1   6.0     2   0.5     4   3.5     5   1.0
4    4    1  15.0     2   9.5     3   9.0     5   4.0
5    5    1  11.0     2   5.5     3   5.0     4   8.5
```

Box 2 Output file from `gsp` command, option 2

```
i  N(i)     distance matrix
---------------------------------
1    1    0.0  1.0  3.0  4.0  1.5
2    2    8.0  0.0  2.0  3.0  0.5
3    3    6.0  0.5  0.0  3.5  1.0
4    4   15.0  9.5  9.0  0.0  4.0
5    5   11.0  5.5  5.0  8.5  0.0
```

7.2.4.3 Transitive Closure

To investigate connections in undirected or directed graphs, a useful concept is the *transitive closure*. Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ denote a (directed) graph. Assume first that the graph is unvalued. The transitive closure of $\mathcal{G}$ is then the graph $\mathcal{G}' = (\mathcal{N}, \mathcal{E}')$ with adjacency matrix $A' = (a'_{ij})$ such that $a'_{ij} = 1$ if $\mathcal{G}$ contains a (directed) path from i to j. If, on the other hand, the graph is valued, a'_{ij} is defined as the value of the shortest path from i to j.

To find the transitive closure of an unvalued graph, one can use the commands `gcon` and/or `gscon` described, respectively, in sections 7.2.3.1 and 7.2.3.2; and if the graph is valued one can use the `gsp` command described in Section 7.2.4.2. All these commands are quite efficient if the graph is sparse. If, however, the graph is dense, it might be more efficient to use an algorithm that directly manipulates the adjacency matrix. An appropriate algorithm for unvalued graphs was invented by Warshall [1962]. The `gtcl` command uses a version of this algorithm discussed by Sedgewick [1990, p. 475]. If the graph is valued, the `gtcl` command uses a version of Warshall's algorithm known as Floyd's algorithm (see, e.g., the discussion in Turau [1996]).

The syntax of the `gtcl` command is shown in the following box. As just indicated, the command can be used with all graph types supported by the `gdd` command.

```
gtcl (
    gn=...,          graph number, def. 1
    nfmt=...,        integer print format, def. 4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. By default, the command uses the first graph (`gn=1`) in the currently defined multigraph. The output file contains the adjacency matrix A' plus two leading columns containing, respectively, the internal and external node numbers. The print format for these additional columns can be specified with the `nfmt` parameter. Note that the

Box 1 Output files created by `gtcl` command

```
undirected, unvalued
--------------------
 1    1    0 1 1 1 1
 2    2    1 0 1 1 1
 3    3    1 1 0 1 1
 4    4    1 1 1 0 1
 5    5    1 1 1 1 0

undirected, valued
--------------------------------
1    1    0.0  1.0  1.5  4.0  1.5
2    2    1.0  0.0  0.5  3.0  0.5
3    3    1.5  0.5  0.0  3.5  1.0
4    4    4.0  3.0  3.5  0.0  3.5
5    5    1.5  0.5  1.0  3.5  0.0

directed, unvalued
--------------------
 1    1    0 1 1 1 1
 2    2    1 0 1 1 1
 3    3    1 1 0 1 1
 4    4    1 1 1 0 1
 5    5    1 1 1 1 0

directed, valued
--------------------------------
1    1    0.0  1.0  3.0  4.0  1.5
2    2    8.0  0.0  2.0  3.0  0.5
3    3    6.0  0.5  0.0  3.5  1.0
4    4   15.0  9.5  9.0  0.0  4.0
5    5   11.0  5.5  5.0  8.5  0.0
```

command requires storage for n^2 bytes (if unvalued) or single precision floating point number (if valued); n being the number of nodes in the graph. Note also that the algorithm sets $a'_{ii} = 1$ if node i has a loop, otherwise $a'_{ii} = 0$.

Example 1 To illustrate the `gtcl` command we use the example data from Section 7.2.4.1. The command file is `gd16.cf`. The `gtcl` command is used with `opt=2`. Box 1 shows output files for four different graph types.

7.2.5 Spanning Trees

Given an undirected and connected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, a *spanning tree* for $\mathcal{G}$ is a graph $\mathcal{G}' = (\mathcal{N}, \mathcal{E}')$ having the same node set, $\mathcal{N}$, but only a subset of the edges, $\mathcal{E}' \subset \mathcal{E}$, such that $\mathcal{G}'$ is a tree. In general, a graph can have several (in fact, a verly large number of different) spanning trees. If the graph is not connected one can calculate a separate spanning tree for each of its connected components and the result set of spanning trees is then called a *forest*. If the graph is valued one can calculate so-called *minimal*, or *maximal*, spanning trees, meaning spanning trees where the sum of edge values is minimal, or maximal.

This section describes commands that can be used to find spanning trees and, in particular, minimum or maximum spanning trees for undirected graphs.

7.2.5.1 Depth-first Spanning Trees describes the `gst` command that finds a depth-first spanning tree.

7.2.5.2 Minimum Spanning Trees describes the `gmst` command that finds a minimum, or maximum, spanning tree.

7.2.5.3 Enumeration of Spanning Trees describes the `gnst` command that enumerates the spanning trees of a graph.

7.2.5.1 Depth-first Spanning Trees

Given an undirected graph, the `gst` command calculates a spanning tree for each connected component of the graph. The algorithm is based on depth-first search. The syntax is shown in the following box.

```
gst(
    gn=...,          graph number, def. 1
    nfmt=...,        integer print format, def. 4
    fmt=...,         print format for values, def. 10.4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify an undirected graph. The output file has four columns. The first column contains an index number for the tree. The other three columns show the edges of the tree (node numbers and edge values).

Example 1 To illustrate we use the graph shown in Figure 1. The data file `gd7.dat` (Box 1) is used to define an undirected graph. The command file is `gd24.cf`. Box 2 shows the output file from the `gst` command.

Box 1 Data file `gd7.dat`

```
 I  J   V
----------
 1  2   1
 2  4   2
 4  5   3
 5  3   4
 3  4   5
 1  3   6
 2  3   7
 6  8   8
 6  7   9
 8  7  10
```

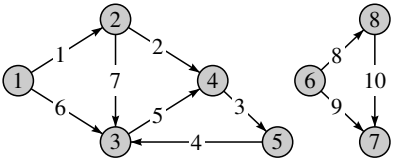


Figure 1 Graph (plotted with `gd23.cf`) of the data shown in Box 1.

Box 2 Output file from `gst` command

```
 T  N(i) N(j)    value
------------------------
 1    1    2     1.0000
 1    2    3     7.0000
 1    3    4     5.0000
 1    4    5     3.0000
 2    6    7     9.0000
 2    7    8    10.0000
```

7.2.5.2 Minimum Spanning Trees

Given an undirected valued graph, the `gmst` command calculates a minimum, or maximum, spanning tree for each of its connected components. The syntax is shown in the following box.

```
gmst (
    alg=...,        algorithm, def. 1
                    1 = Kruskal
                    2 = Prim
    max=...,        1 if maximum spanning tree, def. 0
    gn=...,         graph number, def. 1
    sort,           sort wrt node numbers
    nfmt=...,       integer print format, def. 4
    fmt=...,        print format for values, def. 10.4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify an undirected and valued graph. By default, the command uses the first graph number and calculates a minimum spanning tree. The `max=1` parameter can be used to request a maximum spanning tree.

Two algorithms are available. If `alg=1` (default), the command uses Kruskal's algorithm, if `alg=2` the command uses Prim's algorithm. Both algorithms are described in most books on graph algorithms, see, e.g., Gibbons [1985, ch. 2], Sedgewick [1990, ch. 31], Neumann and Morlock [1993, pp. 199–201]. In general, Kruskal's algorithm is more efficient if the graph is sparse. Both algorithms are applied separately for each component of the graph. Loops (elements in the main diagonal of the adjacency matrix) are not used. Isolated nodes are treated as separate components.

The output file has four columns. The first column contains an index number for the tree. The other three columns show the edges of the tree (node numbers and edge values). By default, when using Kruskal's method, the records are sorted according to edge values, and may have any order when using Prim's method. As an option one can use the `sort`

Box 1 Output file from `gmst` command

```
 T  N(i) N(j)    value
-----------------------
 1    1    2     1.0000
 1    2    4     2.0000
 1    4    5     3.0000
 1    3    5     4.0000
 2    6    8     8.0000
 2    6    7     9.0000
```

parameter to request sorting with respect to node numbers.

Example 1 To illustrate we use the graph shown in Figure 7.2.5.1-1. The data file `gd7.dat` is used to define an undirected and valued graph. The command file is `gd25.cf`. Box 1 shows the output file from the `gmst` command.

7.2.5.3 Enumeration of Spanning Trees

Given an undirected graph, the `gnst` enumerates all of its spanning trees. The algorithm is adapted from McIlroy [1969] and is applied separately to each connected component of the graph. The syntax of the command is shown in the following box.

```
gnst (
    opt=...,        output option, def. 1
                    1 = one record for each tree
                    2 = edge list
    gn=...,         graph number, def. 1
    nfmt=...,       integer print format, def. 4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify an undirected graph. By default, the command uses the first graph number. To illustrate the output options, we use the graph shown in 1. The corresponding data file is `gd10.dat`, the command file for the example is `gd32.cf`.

1. If `opt=1` (default), the output file contains one record for each tree. The first column numbers the connected components of the graph, the second column numbers the trees. Then follow n columns, n being the number of nodes in the graph. Assume that these columns are labeled by $j = 1, \ldots, n$ and let n_j denote the value of column j (in the current record). If then $n_j > 0$, this indicates that the current tree contains an edge from node j to node n_j. This option is illustrated in the left part of Box 1.

2. If `opt=2`, the output file is organized as an edge list, again separately for each connected component of the graph. Like option 1, the first two columns number, respectively, the components and the trees. Then follow four additional columns. The third column shows the number of edges in the tree, the fourth column provides a counter. Finally, the last two columns describe the edge by its starting and ending nodes. This option

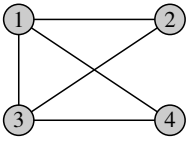


Figure 1 Undirected graph used for examples (`gd31.cf`).

Box 1 Output files from `gnst` command

Option 1

NC	NT	N1	N2	N3	N4
1	1	0	1	1	1
1	2	0	1	1	3
1	3	0	1	2	1
1	4	0	1	2	3
1	5	0	1	4	1
1	6	0	3	1	1
1	7	0	3	1	3
1	8	0	3	4	1

Option 2

NC	NT	edge			
1	1	3	1	2	1
1	1	3	2	3	1
1	1	3	3	4	1
1	2	3	1	2	1
1	2	3	2	3	1
1	2	3	3	4	3
1	3	3	1	2	1
1	3	3	2	3	2
1	3	3	3	4	1
1	4	3	1	2	1
1	4	3	2	3	2
1	4	3	3	4	3
1	5	3	1	2	1
1	5	3	2	3	4
1	5	3	3	4	1
1	6	3	1	2	3
1	6	3	2	3	1
1	6	3	3	4	1
1	7	3	1	2	3
1	7	3	2	3	1
1	7	3	3	4	3
1	8	3	1	2	3
1	8	3	2	3	4
1	8	3	3	4	1

is illustrated in the right part of Box 1.

7.2.6 Cycles

This sections describes commands that can be used to find cycles in undirected and directed graphs.

7.2.6.1 Fundamental Set of Cycles describes the `gcyc` command that finds a fundamental set of cycles for an undirected graph and, optionally, all of its cycles.

7.2.6.2 Enumeration of Cycles describes the `gdcyc` command that can be used to find all cycles in a directed graph.

7.2.6.1 Fundamental Set of Cycles

Given an undirected graph, the `gcyc` command can be used to calculate a fundamental set of cycles and, optionally, all cycles. For a discussion of this notion see, e.g., Gibbons [1985, pp. 54–57]. The algorithm to calculate a fundamental set of cycles is adapted from Paton [1969]. The algorithm to find all cycles, given a fundamental set of cycles, is adapted from Gibbs [1969]. The syntax of the `gcyc` command is shown in the following box.

```
gcyc (
    opt=...,        option, def. 1
                    1 = fundamental cycles, node list
                    2 = fundamental cycles, edge list
                    3 = all cycles, version 1
                    4 = all cycles, version 2
    gn=...,         graph number, def. 1
    nfmt=...,       integer print format, def. 4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify an undirected graph. The data written to the output file depends on the `opt` parameter and will be explained in the example below.

Example 1 To illustrate the `gcyc` command we use the graph shown in Figure 1. (The same example was used by Gibbs [1969] and Paton [1969].) The corresponding data file is `gd9.dat`, the command file for the examples is `gd30.cf`.

1. If `opt=1` (default), the command only calculates a fundamental set of cycles. This is done separately for each of the graph's connected components. Box 1 illustrates the contents of the output file. The first column numbers the components of the graph (in this example, there is only one component). The second column numbers the fundamental cycles, separately for each component. In our example, we find 6 fundamental cycles. The remaining columns describe the corresponding cycle by a list

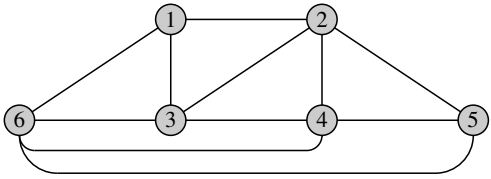


Figure 1 Undirected graph used for examples (`gd29.cf`).

of node numbers. For instance, the first cycle contains the nodes 6, 1, and 3.

2. If `opt=2`, the command again only calculates a fundamental set of cycles but the information is given in a somewhat different format in the output file. This is illustrated in Box 2. Separately for each component of the graph, numbered in the first column, there is one record for each edge in the graph. The edges are numbered in the second column and described, by their starting and ending node, in the third and fourth column. Finally follow m columns, m being the number of fundamental cycles. In each of these columns, a 1 indicates that the corresponding edge is part of the cycle.

3. If `opt=3`, the command calculates all cycles of the graph, based on a previously found fundamental set of cycles. Output from this option is illustrated in Box 3. The first column indicates the connected components of the graph. Then, separately for each component, there is one record for each edge in the graph, numbered in the second column and described by starting and ending nodes in the third and fourth columns. Then follow m columns, m being the number of cycles in the graph. In our example, we find 38 cycles. Each cycle is described by indicators that indicate which of the edges belong to the cycle.

4. Like option 3, if `opt=4` the command calculates all cycles of the graph but provides the information in a somewhat different format. This is illustrated in Box 4. The output file is in two parts. The first part equals the first four columns as described for option 3. Then follows one record for each cycle of the graph that indicates which edges belong to the cycle.

Box 1 Output file from `gcyc` command, option 1

```
NC   idx   node numbers
---------------------------
 1    1    6    1    3
 1    2    5    6    1    2
 1    3    5    6    4
 1    4    4    6    1    2
 1    5    4    6    1    3
 1    6    3    1    2
```

Box 2 Output file from `gcyc` command, option 2

```
NC   idx   edge   indicators
-----------------------------
 1    1    1  2   0 1 0 1 0 1
 1    2    1  3   1 0 0 0 1 1
 1    3    1  6   1 1 0 1 1 0
 1    4    2  3   0 0 0 0 0 1
 1    5    2  4   0 0 0 1 0 0
 1    6    2  5   0 1 0 0 0 0
 1    7    3  4   0 0 0 0 1 0
 1    8    3  6   1 0 0 0 0 0
 1    9    4  5   0 0 1 0 0 0
 1   10    4  6   0 0 1 1 1 0
 1   11    5  6   0 1 1 0 0 0
```

Box 3 Part of output file from `gcyc` command, option 3

```
NC   idx   edge   1        indicators describing cycles        38
-------------------------------------------------------------------
 1    1    1  2   0 1 1 1 1 0 1 0 1 0 ... 0 0 0 0 0 0 0 0 1 0 1 1
 1    2    1  3   1 1 0 1 0 0 1 0 1 0 ... 0 1 0 0 1 1 0 0 0 0 0 1
 1    3    1  6   1 0 1 0 1 0 0 0 0 0 ... 0 1 0 0 1 1 0 0 1 0 1 0
 1    4    2  3   0 0 0 0 0 0 0 0 0 0 ... 1 1 1 1 1 1 1 1 1 1 1 1
 1    5    2  4   0 0 0 0 0 0 0 1 1 1 1 ... 0 0 1 1 1 1 0 0 0 1 0 0
 1    6    2  5   0 1 1 1 1 0 0 1 0 1 ... 1 1 0 0 0 0 0 1 1 0 0 0 0
 1    7    3  4   0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 1 1 1 1 1 0
 1    8    3  6   1 1 0 1 0 0 1 0 1 0 ... 1 0 1 1 0 0 0 0 0 0 0 0 0
 1    9    4  5   0 0 0 1 1 1 0 0 1 1 ... 1 1 0 1 1 0 0 1 1 0 0 0
 1   10    4  6   0 0 0 1 1 1 1 1 0 0 ... 1 1 1 0 0 1 1 0 0 0 1 0
 1   11    5  6   0 1 1 0 0 1 0 1 1 0 ... 0 0 0 1 1 0 1 0 1 0 0 0
```

Box 4 Output file from gcyc command, option 4

```
NC   idx     edge
-----------------
 1     1     1     2
 1     2     1     3
 1     3     1     6
 1     4     2     3
 1     5     2     4
 1     6     2     5
 1     7     3     4
 1     8     3     6
 1     9     4     5
 1    10     4     6
 1    11     5     6
 1     1    0 1 1 0 0 0 0 1 0 0 0
 1     2    1 1 0 0 0 1 0 1 0 0 1
 1     3    1 0 1 0 0 1 0 0 0 0 1
 1     4    1 1 0 0 0 1 0 1 1 1 0
 1     5    1 0 1 0 0 1 0 0 1 1 0
 1     6    0 0 0 0 0 0 0 0 1 1 1
 1     7    1 1 0 0 1 0 0 1 0 1 0
 1     8    0 0 0 0 1 1 0 0 0 1 1
 1     9    1 1 0 0 1 0 0 1 1 0 1
 1    10    0 0 0 0 1 1 0 0 1 0 0
 1    11    1 0 1 0 1 0 0 0 1 0 1
 1    12    1 0 1 0 1 0 0 0 0 1 0
 1    13    0 0 0 0 0 0 1 1 0 1 0
 1    14    1 1 0 0 0 1 1 0 0 1 1
 1    15    0 0 0 0 0 0 1 1 1 0 1
 1    16    1 0 1 0 0 1 1 1 1 0 0
 1    17    1 1 0 0 0 1 1 0 1 0 0
 1    18    0 1 1 0 0 0 1 0 1 0 1
 1    19    1 0 1 0 1 0 1 1 0 0 0
 1    20    0 0 0 0 1 1 1 1 0 0 1
 1    21    0 1 1 0 1 1 1 0 0 0 1
 1    22    1 1 0 0 1 0 1 0 0 0 0
 1    23    0 1 1 0 0 0 1 0 0 1 0
 1    24    1 0 1 1 0 0 0 1 0 0 0
 1    25    0 0 0 1 0 1 0 1 0 0 1
 1    26    0 1 1 1 0 1 0 0 0 0 1
 1    27    0 0 0 1 0 1 0 1 1 1 0
 1    28    0 1 1 1 0 1 0 0 1 1 0
 1    29    0 0 0 1 1 0 0 1 0 1 0
 1    30    0 0 0 1 1 0 0 1 1 0 1
 1    31    0 1 1 1 1 0 0 0 1 0 1
 1    32    0 1 1 1 1 0 0 0 0 1 0
 1    33    0 0 0 1 0 1 1 0 0 1 1
 1    34    0 0 0 1 0 1 1 0 1 0 0
 1    35    1 0 1 1 0 0 1 0 1 0 1
 1    36    0 0 0 1 1 0 1 0 0 0 0
 1    37    1 0 1 1 0 0 1 0 0 1 0
 1    38    1 1 0 1 0 0 0 0 0 0 0
```

7.2.6.2 Enumeration of Cycles

The notion of a fundamental set of cycles makes sense only for undirected graphs. An efficient algorithm to find all cycles in directed graphs was developed by Tiernan [1970]. The `gdcyc` command is based on this algorithm. It requires a directed graph defined with option 1 (edge list) of the `gdd` command. The syntax is shown in the following box.

```
gdcyc (
    opt=...,        output option, def. 1
                    1 = one record for each cycle
                    2 = one record for each node
    gn=...,         graph number, def. 1
    nfmt=...,       integer print format, def. 4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. In order to illustrate the two output options we use the graph from Section 7.2.4.1. The command file is `gd33.cf`. The output from option 1 is shown in the left part of Box 1. There is one record for each cycle, numbered in the first column of the output file. Each cycle is described by a sequence of node numbers. The length of the cycle is shown in the second column. Then follow the node numbers contained in the cycle. The output from option 2 is illustrated in the right part of Box 1.

Box 1 Output files from `gdcyc` command

Option 1

C	N	node numbers				
1	3	1	2	3		
2	5	1	2	4	5	3
3	4	1	2	5	3	
4	2	2	3			
5	4	2	4	5	3	
6	3	2	5	3		
7	3	3	4	5		

Option 2

C	N	node
1	3	1
1	3	2
1	3	3
2	5	1
2	5	2
2	5	4
2	5	5
2	5	3
3	4	1
3	4	2
3	4	5
3	4	3
4	2	2
4	2	3
5	4	2
5	4	4
5	4	5
5	4	3
6	3	2
6	3	5
6	3	3
7	3	3
7	3	4
7	3	5

7.2.7 Eigenvalues and Eigenvectors

The `gev` command can be used to calculated a selected number of eigenvalues and eigenvectors of the adjacency matrix of an undirected graph. The algorithm is adapted from Nikolai [1979] and, since storage of the complete adjacency matrix is not required, can be used efficiently with large sparse graphs defined with `gdd` option 1. The syntax of the command is shown in the following box.

```
gev (
    n=...,          number of eigenvalues/vectors, def. 1
    eps=...,        required accuracy, def. 1.e-6
    mxit=...,       max number of iterations, def. 30
    gn=...,         graph number, def. 1
    fmt=...,        print format, def. 10.4
    prot=...,       protocol file
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify an undirected graph. The `n` parameter specifies how many eigenvalues/vectors will be calculated. The command tries to find those eigenvalues (and corresponding eigenvectors) whose absolute value is largest. The maximal number of eigenvalues/vectors that can be requested is $N - 1$, where N is the number of nodes in the graph. The calculated eigenvalues are shown in the standard output, the corresponding eigenvectors are written into the output file.

1. Note that the command might fail to find the required number of eigenvalues/vectors. This can happen, in particular, if the eigenvalues are not well separated. As a remedy, one can then try to request a calculation of *more* eigenvalues than are actually required.

2. To achieve the required accuracy it may be necessary to allow for a larger number of iterations. The standard output reports how many iterations have been performed. If this number equals the maximal number of iterations specified with the `mxit` parameter, one should

specify a higher value.

3. There is no direct relationship between the accuracy specified with the `eps` parameter and the number of correct digits in the calculated eigenvalues/vectors. Information about the actually achieved accuracy can be found in a protocol file that can be requested with the `prot` parameter. For each eigenvalue λ, and corresponding eigenvector x, the protocol file shows the value of

$$\max_j \{ \, | \, (Ax)_j - \lambda x_j \, | \, \}$$

The TDA example archive contains a command file, `gd28.cf`, that illustrates how to use the `gev` command and compares its result with that of the `mevs` matrix command.

7.2.8 Flows

This section describes commands that can be used to analyze flows in directed graphs and, in particular, to find maximal flows.

7.2.8.1 Maximal Flows describes the `gflow` command that finds maximal flows in directed graphs.

7.2.8.1 Maximal Flows

Given a directed graph, the `gflow` command finds a maximal flow for each pair of nodes that is connected by at least one directed path. If the graph is unvalued, it is assumed that all edges have value 1. The `gflow` command uses an algorithm first developed by Edmonds and Karp [1972]; the implementation is based on its discussion in Turau [1996, pp. 171–179]. The syntax of the `gflow` command is shown in the following box.

```
gflow (
    opt=...,        output option, def. 1
                    1 = one record per flow
                    2 = square flow matrix
    gn=...,         graph number, def. 1
    nfmt=...,       integer print format, def. 4
    fmt=...,        print format for flow values, def. 10.4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify a directed graph that must be defined with option 1 of the `gdd` command, i.e. as an edge list.

Example 1 To illustrate the output options we use the graph shown in Figure 1. (The example is taken from Turau [1996, p. 164].) The corresponding data file is `gd11.dat`, the command file for the examples is `gd35.cf`.

1. If `opt=1` (default), the output file contains one record for each pair of nodes that is connected by at least one directed path. The first column counts the records. Then follow four columns containing, respectively, the internal and external node numbers of the source and the sink. The final column shows the value of the maximal flow. This option is illustrated in Box 1.

2. If `opt=2`, the output file contains an (n, n) square matrix, n being

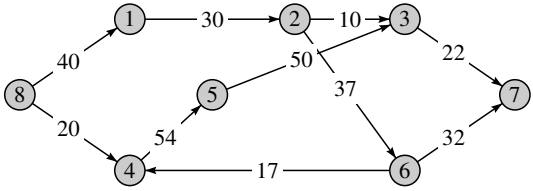


Figure 1 Directed graph used for examples (`gd34.cf`).

the number of nodes in the graph, plus two leading columns containing, respectively, the internal and external node numbers. The (i, j) entry of the matrix contains the value of a maximal flow from node i to node j, or -1 if there is no directed path from i to j. This option is illustrated in Box 2.

Box 1 Output file from `gflow` command, option 1

Idx	i	j	N(i)	N(j)	flow
1	1	2	1	2	30
2	1	3	1	3	27
3	1	4	1	4	17
4	1	5	1	5	17
5	1	6	1	6	30
6	1	7	1	7	30
7	2	3	2	3	27
8	2	4	2	4	17
9	2	5	2	5	17
10	2	6	2	6	37
11	2	7	2	7	47
12	3	7	3	7	22
13	4	3	4	3	50
14	4	5	4	5	54
15	4	7	4	7	22
16	5	3	5	3	50
17	5	7	5	7	22
18	6	3	6	3	17
19	6	4	6	4	17
20	6	5	6	5	17
21	6	7	6	7	49
22	8	1	8	1	40
23	8	2	8	2	30
24	8	3	8	3	47
25	8	4	8	4	37
26	8	5	8	5	37
27	8	6	8	6	30
28	8	7	8	7	50

Box 2 Output file from `gflow` command, option 2

i	N(i)								
1	1	0	30	27	17	17	30	30	-1
2	2	-1	0	27	17	17	37	47	-1
3	3	-1	-1	0	-1	-1	-1	22	-1
4	4	-1	-1	50	0	54	-1	22	-1
5	5	-1	-1	50	-1	0	-1	22	-1
6	6	-1	-1	17	17	17	0	49	-1
7	7	-1	-1	-1	-1	-1	-1	0	-1
8	8	40	30	47	37	37	30	50	0

7.2.9 Subgroups

This section describes commands that find specific kinds of subgroups of the nodes of a graph.

7.2.9.1 Cliques describes a command that finds the cliques (maximal complete subgraphs) of a graph.

7.2.9.2 Compact Sets describes a command that finds the compact sets of a graph.

7.2.9.3 Independent Sets describes a command that tries to find maximal independent sets of a graph.

7.2.9.1 Cliques

Given an undirected graph, one can consider its maximally connected subgraphs, also called *cliques*. To find the cliques in a graph, one can use the *gcliq* command with syntax shown in the following box.

```
gcliq(
    alg=...,          algorithm, def. 1
                      1 = Bron-Kerbosch
                      2 = Harary-Ross
    gn=...,           graph number, def. 1
    nfmt=...,         integer print format, def. 4
    min=...,          minimum size of cliques, def. 3
    sort,             sort node numbers
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify an undirected graph. There are two algorithms. If `alg=1` (default), the command uses an algorithm developed by Bron and Kerbosch [1973]. If `alg=2` the command uses an algorithm proposed by Harary and Ross [1957]. The algorithms are applied separately to each connected component of the graph.

The output file will contain one record for each clique found in the graph. The first column counts the connected components, the second column counts the cliques that are found in the component. The third column shows the number of nodes in the clique, finally follows the list of external node numbers of the nodes contained in the clique. As an option one can use the `sort` parameter to request that these node numbers are written in ascending order. As a further option, one can use the `min` parameter to specify a minimum size of the cliques to be written into the output file. The minimum size must not be smaller than 2 if using algorithm 1, and not smaller than 3 if using algorithm 2.

Example 1 To illustrate the `gcliq` command we use the graph shown in Figure 1. The corresponding data file is `gd16.dat`, the command file

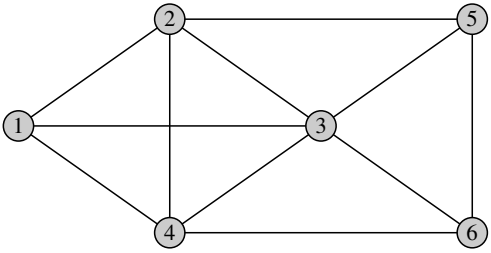


Figure 1 Undirected graph for examples (`gd43.cf`).

for the examples is `gd42.cf`. The upper part of Box 1 shows the output file if `alg=1`, the lower part shows the output file if `alg=2`. The columns show, respectively: the component number (C), the number of the clique (NC), the number of nodes in the clique (N), and finally the external node numbers of the nodes contained in the clique. Notice that there is a bug in our implementation of the Harary-Ross algorithm: it is not always able to detect that a clique is actually part of a larger clique and should not be written, therefore, into the output file. We allow for this bug in order to save internal memory and overhead.

Box 1 Output files from `gcliq` command

```
Algorithm 1

 C   NC    N    list of node numbers
-----------------------------------
 1    1    4    3    2    1    4
 1    2    3    3    2    5
 1    3    3    3    6    4
 1    4    3    3    6    5

Algorithm 2

 C   NC    N    list of node numbers
-----------------------------------
 1    1    4    1    2    3    4
 1    2    3    4    2    3
 1    3    3    3    2    5
 1    4    3    4    3    6
 1    5    3    3    5    6
```

7.2.9.2 Compact Sets

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, v)$ denote an undirected valued graph with n nodes. A subset $C \subset \mathcal{N}$ is called a *compact set* if

$$1 < |\,C\,| < n, \text{ and}$$
$$\max\{\, v(i,j) \,|\, i,j \in C \,\} < \min\{\, v(i,j) \,|\, i \in C, j \in \mathcal{N} - C \,\}$$

A graph can have none, one, or more than one compact set, the maximum is $n-2$. If there are two or more compact sets, they are disjoint or nested.

To find the compact sets of an undirected valued graph we use an algorithm developed by Liang [1993]. The command is `gcset` with syntax shown in the following box.

```
gcset (
    opt=...,        output option, def. 1
                    1 = one record for each compact set
                    2 = one record for each node
    alg=...,        MST algorithm, def. 1
                    1 = Kruskal
                    2 = Prim
    gn=...,         graph number, def. 1
    nfmt=...,       integer print format, def. 4
    fmt=...,        print format for flow values, def. 10.4
    df=...,         print pseudo image graph
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify an undirected valued graph. Since the algorithm is based on first creating a minimum spanning tree of the graph, one can use the `alg` parameter to select Kruskal's or Prim's algorithm, see Section 7.2.5.2. Notice that the algorithm is applied to each connected component of the graph separately.

Example 1 To illustrate the `gcset` command we use the graph shown in Figure 1. (The example is taken from Liang [1993].) The corresponding data file is `gd13.dat`, the command file for the examples is `gd39.cf`.

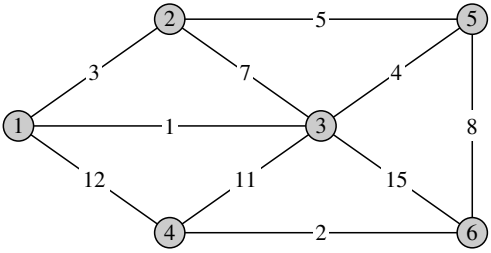


Figure 1 Undirected valued graph for examples (`gd38.cf`).

The upper part of Box 1 shows the output file if `opt=1` (default). The first column numbers the connected components of the graph. In our example, there is only a single component. The second column counts the compact sets found in the component. In this example, we find three compact sets, $\{1,3\}$, $\{4,6\}$, and $\{1,2,3,5\}$. The third column of the output file shows their size, followed by their diameter, defined by $\max\{\, v(i,j) \,|\, i,j \in C \,\}$, and the minimal distance, $\min\{\, v(i,j) \,|\, i \in C, j \in \mathcal{N} - C \,\}$. The same information is given if `opt=2` but the output file will then contain one record for each node. Finally, if the `df` parameter is used, the command creates an additional output file as shown in the lower part of Box 1. Its contents correspond to an edge list for a reduced graph whose nodes are the compact sets, or single nodes, of the original graph. The columns labeled `i` and `j` show the internal node numbers, the columns labeled `N(i)` and `N(j)` show the corresponding external node numbers, respectively. If one of the nodes is a compact set, indicated by its size in column 5 or column 6, the set is represented by its lowest node number. Finally, the last column contains the minimal distance between the nodes in the reduced graph.

Box 1 Output files from `gcset` command

```
Option 1

 C   NC    S   diameter   distance    nodes
-------------------------------------------------------
 1    1    2     1.0000     3.0000    1    3
 1    2    2     2.0000     8.0000    4    6
 1    3    4     7.0000     8.0000    1    2    3    5

Option 2

 1    1    2     1.0000     3.0000    1
 1    1    2     1.0000     3.0000    3
 1    2    2     2.0000     8.0000    4
 1    2    2     2.0000     8.0000    6
 1    3    4     7.0000     8.0000    1
 1    3    4     7.0000     8.0000    2
 1    3    4     7.0000     8.0000    3
 1    3    4     7.0000     8.0000    5

Output from df parameter

 i    j  N(i) N(j)  S1   S2   distance
-----------------------------------------
 1    4    1    4    4    2     8.0000
```

7.2.9.3 Independent Sets

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ denote an undirected graph. A subset $U \subset \mathcal{N}$ is called an *independent set* if for all $i, j \in U$ there isn't an edge (i, j) in $\mathcal{E}$. U is called *maximal* if when adding a node, U is no longer an independent set. An independent set that has maximal cardinality is called a *maximum independent set*.

In order to find a maximum independent set one would need an enumerative procedure, or some variant, and this will only be feasible for very small graphs. We therefore use an algorithm developed by Feo, Resende and Smith [1994] that is based on a randomized search procedure. The command is `giset` with syntax shown in the following box.

```
giset (
    mxit=...,       max number of iterations, def. 100
    idf=...,        values for α and β, def. 0.1,0.1
    seed=...,       seed of random number generator,
                    def. 270001
    min=...,        minimal size of independent set
    gn=...,         graph number, def. 1
    cg=...,         1 to use complementary graph, def. 0
    nfmt=...,       integer print format, def. 4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify an undirected graph. By default, the algorithm tries to find a maximum independent set and uses the maximum number of iterations specified with the `mxit` parameter. The best independent set found in these iterations is finally reported and written into the output file. If the `min` parameter is used to specify a required minimum size for the independent set, iterations are stopped as soon as an independent set having the required size is found. If `silent = -1`, the command reports about its iterations on the standard error output.

For its randomized search procedure, the algorithm uses a random number generator proposed by Schrage [1979]. One possibility to control

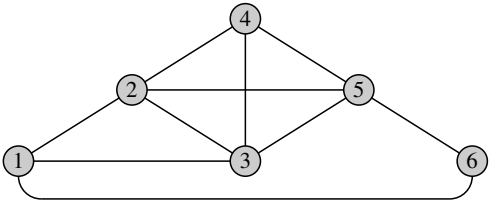


Figure 1 Undirected graph used for examples (`gd36.cf`).

the algorithm is to select different seeds with the `seed` parameter (must be a positive integer). Further control is possible with the parameter `idf =` α, β where $0 < \alpha, \beta \leq 1$. For a full description of these (and some additional) parameters, see Feo et al. [1994]. (Since our implementation of the algorithm is preliminary, we haven't made available all of its parameters in the `giset` command.)

A further option is provided by the `cg` parameter. By default, the `giset` command searches for a maximum independent set in the graph defined by the current `gdd` data structure. The `cg=1` parameter can be used to request that the search is done in the complementary graph, meaning a graph $\mathcal{G}_c = (\mathcal{N}, \mathcal{E}_c)$ where $\mathcal{E}_c$ is the complement of $\mathcal{E}$. A maximum independent set in this complementary graph will then provide a maximal clique (maximal complete subgraph) of the original graph.

The resulting output file contains one record for each node being a member of the independent set. There are three columns. Column 1 counts the records, columns 2 and 3 contain, respectively, the internal and external node numbers.

Example 1 To illustrate the `giset` command we use the graph shown in Figure 1. The corresponding data file is `gd12.dat`, the command file for the examples is `gd37.cf`. With default parameters the `giset` command finds the independent set $\{3, 6\}$. Using the `cg=1` parameter to search for a maximum independent set in the complementary graph, we find $\{2, 3, 4, 5\}$ corresponding to the largest clique in the original graph.

7.3 Combinatorial Optimization

This section describes a variety of commands which, more or less, refer to problems of combinatorial optimization. Subsections are as follows.

7.3.1 Assignment Problems describes commands for simple and quadratic assignment problems.

7.3.1 Assignment Problems

This section describes commands for solving simple and quadratic assignment problems. Subsections are as follows.

7.3.1.1 Column Permutations describes a command that finds optimal permutations of matrix columns.

7.3.1.2 Quadratic Assignment describes a command that finds solutions of the quadratic assignment problems.

7.3.1.1 Column Permutations

Given an (n, n) matrix $D = (d_{ij})$, a simple assignment problem consists in finding a permutation of column indices, say π, such that

$$\sum_{i=1}^{n} d_{i,\pi(i)} \longrightarrow \min$$

To solve this problem, we use an algorithm developed by Carpaneto and Toth [1980]. The command is `gap` with syntax shown in the following box.

```
gap (
    gn=...,          graph number, def. 1
    nfmt=...,        integer print format, def. 4
    df=...,          write permuted matrix
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The command requires as input a graph defined with the `gdd` command. It is expected that edge values are integral. In any case, edge values will be casted into integer values without warning. By default, the command only writes the optimal permutation vector into its standard output file. As an option, one can use the `df` parameter to request an output file containing the permuted matrix.

Example 1 To illustrate the `gap` command we use the data shown in Box 1. The data file is `co1.dat`, the command file for the example is `co1.cf`. Output files are shown in Box 2.

Box 1 Example data file `co1.dat`

```
60    0    0   76    0    0
 0   40   18    0   60   24
60   16    2    4    0   40
 0   27   18    3   55   75
 0   40   62   16   11   53
28    4   10   84    0   16
```

Box 2 Output files from `gap` command

```
standard output file

i  p(i)
-------
1    6
2    4
3    3
4    1
5    5
6    2

second output file (permuted matrix)
----------------------------
 0   76    0   60    0    0
24    0   18    0   60   40
40    4    2   60    0   16
75    3   18    0   55   27
53   16   62    0   11   40
16   84   10   28    0    4
```

7.3.1.2 Quadratic Assignment

Given two integer-valued and symmetric (n, n) matrices, $D = (d_{ij})$ and $F = (f_{ij})$, the quadratic assignment problem consists in finding a permutation, say π, such that

$$\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} f_{\pi(i),\pi(j)} \longrightarrow \min$$

Finding exact solutions is only possible for small matrix dimensions. To solve the problem for large matrices, we use an algorithm developed by Resende, Pardalos, and Li [1996]. The algorithm uses a greedy, randomized, adaptive search procedure (GRASP) in order to find an approximative solution.

The command is `gqap` with syntax shown in the following box. All parameters except for an output file to be specified on the right-hand side are optional.

```
gqap(
    alg=...,        algorithm, def. 1
                    1 = approximative solution
    mxit=...,       max number of iterations, def. 100
    seed=...,       seed of random number generator,
                    def. 270001
    idf=...,        specification of α and β, def. idf=0.25,0.5
    gn=...,         graph number for D and F
    df=...,         write permuted matrix
    nfmt=...,       integer print format, def. 4
) = fname;
```

The command expects two integer-valued graphs whose graph numbers must be specified with the `gn` parameter. By default, the command uses the first two graphs of the current multigraph. The algorithm always performs the maximum number of iterations specified with the `mxit` parameter. Further control is possible with the `seed` parameter that specifies a seed for the random number generator (Schrage [1979]), and

Box 1 Example data file `co2.dat`

```
 F  D
------
 1  5
 7  3
 2  1
```

Box 2 Output files from `gqap` command

```
standard output file

i  p(i)
-------
1    2
2    1
3    3

second output file (permuted multigraph)

i    j  N(i) N(j)   F    D
---------------------------
1    1    1    1    0    0
1    2    1    2    1    5
1    3    1    3    2    3
2    1    2    1    1    5
2    2    2    2    0    0
2    3    2    3    7    1
3    1    3    1    2    3
3    2    3    2    7    1
3    3    3    3    0    0
```

with the `idf` parameter that can be used to specify values for α and β ($0 < \alpha, \beta \leq 1$) for the GRASP procedure.

Example 1 To illustrate the `gqap` command we use the data shown in Box 1. The data file is `co2.dat` and contains the lower triangles of the F and D matrices, respectively. (The multigraph is defined with option 3 of the `gdd` command.) The command file for the example is `co2.cf`. Output files are shown in Box 2. The best cost value found is 36.

7.4 Representation of Proximities

This part of the manual deals with methods for the representation of proximities between a set of objects. We shall use the following terminology and notation. $\Omega = \{\omega_1, \ldots, \omega_N\}$ is a set of N objects. A proximity index d is a mapping of $\Omega \times \Omega$ into the set of nonnegative real numbers.

1. The proximity index d is called a *pseudo-distance*, if it is symmetric, that is, $d(\omega_i, \omega_j) = d(\omega_j, \omega_i)$ for all $\omega_i, \omega_j \in \Omega$.

2. A pseudo-distance d is called a *dissimilarity*, if $d(\omega_i, \omega_j) = 0$ implies $\omega_i = \omega_j$.

3. A dissimilarity index d is called a *distance*, if we have in addition the triangle inequality, that is, $d(\omega_i, \omega_k) \leq d(\omega_i, \omega_j) + d(\omega_j, \omega_k)$, for all

Ω is called a metric space if d is a distance, it is sometimes called a premetric space if d is a pseudo-distance or dissimilarity.

There is obviously a close connection between the notion of proximities and the concept of an undirected valued graph, say $\mathcal{G} = (\mathcal{N}, \mathcal{E}, v)$. We can assume that each node of the graph represents one of the objects and $v(i, j)$ provides the proximity between objects i and j. Consequently, we will assume that our input data are given by a relational data structure that defines a valued graph. Most procedures also assume that the graph is undirected corresponding to a symmetric proximity matrix and that edge values can be interpreted as providing (at least) a pseudo-distance. We do not require that the graph is complete. Following our general conventions about relational data, only non-negative edge values define valid edges.

As a useful side effect of the assumption that proximities are given by a valued graph, we get a clear distinction between two essentially different questions: (1) How to define proximities for a set of objects, and (2) how to represent the proximities. All procedures discussed in this section of the manual assume that the first question has been settled in some way and focus solely on the second question. There remains, of course, a somewhat obscure borderline to procedures commonly discussed under the heading of "cluster analysis". However, we shall not try to find a

clear distinction between "representation" and "clustering". Some standard clustering procedures will be discussed in a separate section of the manual. Subsections in this part are as follows.

7.4.2 Spectrum-based Projections deals with projection methods that use the eigenvalues and eigenvectors of a proximity matrix.

7.4.4 Comparing Configurations deals with the question how to compare representations of a proximity matrix.

7.4.5 Tree Representations describes approaches that use trees to represent a proximity matrix.

7.4.2 Spectrum-based Projections

This section deals with projection methods for proximity matrices based on their eigenvalues and eigenvectors. The subsections are as follows.

7.4.2.1 Direct Projection of Proximities

We assume as given an (n, n) proximity matrix, $D = (d_{ij})$, such that d_{ij} can be interpreted as a dissimilarity index for objects i and j. We can then view the ith row of D as a representation of the location of object i with respect to all other objects. On the other hand, we can also think of the rows of D as points in an n-dimensional space. This view leads to the question whether it may make sense to construct a projection of D onto a lower dimensional space, most often a plane, that can provide some information about the structure of D.

This section describes one possibility to construct such a projection, directly based on D. We shall therefore assume that D is complete, meaning there is a valid dissimilarity index $d_{ij} \geq 0$, for all pairs of objects, i and J. Or otherwise, that missing values can sensibly be substituted by a zero proximity. Since we are finally interested in a graphical display, we restrict the discussion to projections onto a plane.

We begin with a short reminder of the notion of projection, for a detailed discussion see, e.g., Anton and Rorres [1991, Sec. 5.3]. In doing this we refer to n-dimensional Euclidean space, $\mathbf{R}^n$. If u is any vector in $\mathbf{R}^n$, it is written as a column vector $u = (u_1, \ldots, u_n)'$, with $u_1, \ldots, u_n$ the standard coordinates of u, that is, its coordinates with respect to the standard basis of $\mathbf{R}^n$ that is given by the column vectors in the identity matrix I_n. Of course, instead of the standard basis, I_n, one can represent the vectors in $\mathbf{R}^n$ by any other basis, that is, a set of n independent vectors. It is particularly convenient to use an orthonormal basis, that is, a set of n vectors which are mutually orthogonal and have unit length. It follows that the columns of an (n, n) matrix V form an orthonormal basis for $\mathbf{R}^n$ if V is an orthogonal matrix, that is

$$V'V = VV' = I_n$$

If we have such an orthonormal basis, say V, we can represent any vector u in $\mathbf{R}^n$ with respect to this (new) basis by $w = V'u$. The vector w represents the coordinates of u with respect to V. The standard coordinates of u are then given by $u = Vw$.

Now, let W be a two-dimensional subspace of $\mathbf{R}^n$, and let a basis for this plane be given by two linear independent vectors w^1 and w^2. This implies that any vector (point) in W can be written as a linear

combination of w^1 and w^2. We can then use the following theorem of linear algebra (see, e.g., Anton and Rorres [1991, p. 239]):

If u is any vector in $\mathbf{R}^n$, then:

1. u can be expressed in exactly one way as $u = a + b$, where a and b are also vectors in $\mathbf{R}^n$ and have the following properties:

   a is contained in W, that is, a can be written as $a = a_1 w^1 + a_2 w^2$;[1] and b is orthogonal to W, that is, b is orthogonal to all vectors in W.[2] It follows that $u'u = a'a + b'b$.

   The vector a is called the *orthogonal projection* of u on W; and b is called the *component of u orthogonal to W*.

2. If $\{w^1, w^2\}$ is an orthonormal basis for W, then the coordinates of a with respect to this basis are given by

$$a = (u'w^1)w^1 + (u'w^2)w^2 \tag{1}$$

This result can be used to find a projection of the rows of the proximity matrix D onto a plane which is, in a sense, optimal, meaning that the projections onto the plane have a maximal length and the components that are orthogonal to the plane have minimal length. The calculation begins with a singular value decomposition of D. We then have

$$D = U\Lambda V'$$

with orthogonal (n, n) matrices U and V, and a diagonal (n, n) matrix Λ containing the singular values, $\lambda_1, \ldots, \lambda_n$. Since V is orthogonal, it immediately provides an orthonormal basis for $\mathbf{R}^m$, m being the rank of V and equals the rank of D.[3] Defining then

$$H = DV = U\Lambda$$

we see that the rows of H contain the coordinates of the rows of D with respect to the orthonormal basis V.

---

[1] Note that a_1 and a_2 are scalars, the coordinates of a with respect to the basis $\{w^1, w^2\}$ of W; w^1 and w^2 are, of course, vectors.

[2] In particular, b is orthogonal to w^1 ($b'w^1 = 0$) and to w^2 ($b'w^2 = 0$); it then immediately follows that b is orthogonal to all vectors in W.

[3] Of course, $\text{rank}(D) \leq n - 1$. However, since we are finally only interested in projections onto a plane, it suffices to assume that m, the rank of D, is at least 2.

Now, take any two columns of V, say v_r and v_s, to define a two-dimensional subspace, i.e. a plane for the projection, and let d_i be the ith row of D (written as a column vector). By the theorem given above, d_i can be expressed as $d_i = p_i + q_i$ with p_i the orthogonal projection of d_i onto the plane spanned by v_r and v_s, and q_i the component of d_i that is orthogonal to this plane. We want to maximize the length of the projections, p_i, for all rows of D, that is

$$\Sigma_i \parallel p_i \parallel^2 = \Sigma_i p_i' p_i \rightarrow \max$$

As seen by (1), the coordinates of p_i with respect to V are given by the rows of Dv_r (first coordinate) and Dv_s (second coordinate).[4] Consequently,

$$\begin{aligned}\Sigma_i p_i' p_i &= v_r' D' D v_r + v_s' D' D v_s \\ &= \lambda_r u_r' u_r \lambda_r + \lambda_s u_s' u_s \lambda_s \\ &= \lambda_r^2 + \lambda_s^2\end{aligned}$$

since u_r and u_s are columns of the orthogonal matrix U ($U'U = I_n$). The result is that, to find an optimal plane to project the rows of D, one should use the columns of V, say v_r and v_s, which correspond to the two largest singular values of D. So the coordinates to be used for a two-dimensional display of the structure of D should be Dv_r and Dv_s. Of course, the quality of the projection depends on the spectrum of D. Some indication is given by the relative size of the singular values which can be summarized by the following indicator:

$$\sqrt{(\lambda_r^2 + \lambda_s^2) / \Sigma_{t,l} \lambda_{t,l}^2} \tag{2}$$

The nominator equals the total length of the projected rows of A, and the denominator equals their total length before projection.

[4] Explicitly written, $p_i = (d_i' v_r, d_i' v_s)'$.

7.4.2.2 Spectrum-based Configurations

We assume as given a complete (n, n) proximity matrix, $D = (d_{ij})$, such that d_{ij} can be interpreted as a dissimilarity index for objects i and j. Instead of directly projecting D onto a plane, as was discussed in Section 7.4.2.1, one can look for a configuration of n points in a metric space such that their metric distances in some sense optimally approximate the given proximities. There are different approaches to this problem. Here we discuss what is sometimes called *classical metric scaling* (Torgerson [1965]). Starting point is the definition of an (n, n) matrix $A = (a_{ij})$ by

$$a_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{i.}^2 - d_{.j}^2 + d_{..}^2) \tag{3}$$

where

$$d_{i.}^2 = \sum_{j=1}^{n} d_{ij}^2/n\,, \quad d_{.j}^2 = \sum_{i=1}^{n} d_{ij}^2/n\,, \quad d_{..}^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^2/n^2$$

Since D is symmetric, also A is symmetric; and all row and column sums of A are zero, implying that $\text{rank}(A) \leq n - 1$. The proximities d_{ij} can be recovered by using the formula

$$d_{ij}^2 = a_{ii} - a_{ij} - a_{ji} + a_{jj} \tag{4}$$

Now, using this matrix A, one can formulate the following theorem: If, and only if, A is positive semi-definite, there is a dimensionality m and a set of n vectors $x_1, \ldots, x_n \in \mathbf{R}^m$ so that the Euclidean distances $d(x_i, x_j)$ equal the proximities d_{ij} that have been used to define A. For a proof see, e.g., Falk et al. [1995, p. 269]).

Let us then assume that A is positive semi-definite, with $\text{rank}(A) = m < n$. (What to do if A is not positive semi-definite will be discussed below.) Since A is also symmetric, it has exactly m positive real eigenvalues, and the remaining $n - m$ eigenvalues are zero.[1] We can assume

[1] Properties of symmetric matrices are discussed in most linear algebra texts; see, e.g., Anton and Rorres [1991]. Here are some of the basic facts. Let A be a symmetric (n, n) matrix. Then: (1) All eigenvalues of A are real, and the rank of A equals the number of non-zero eigenvalues. (2) The system of eigenvectors of A, say R, can be chosen to be orthogonal, that is, $R'R = RR' = I$. (3) A is orthogonally diagonalizable, that is, $A = R\Lambda R'$, with Λ the diagonal matrix of eigenvalues of A. (4) A is positive definite [semi-definite] if, and only if, $\lambda > 0$ [$\lambda \geq 0$] for all eigenvalues of A.

that these eigenvalues are in descending order: $\lambda_1 \geq \cdots \geq \lambda_n$. Let $\Lambda = \text{diag}\{\lambda_1, \ldots, \lambda_n\}$ denote the diagonal matrix containing the eigenvalues, and let R denote the (n, n) matrix of the corresponding (column) eigenvectors. We then have

$$AR = R\Lambda$$

Furthermore, since A is symmetric and positive semi-definite, the eigenvectors can be normalized in such a way that R becomes an orthogonal matrix, that is, $RR' = I$. It follows that A can be expressed as

$$A = R\Lambda R' = ZZ' \quad \text{with} \quad Z = R\Lambda^{1/2}$$

Finally, since there are only m positive eigenvalues and $\lambda_{m+1}, \ldots, \lambda_n = 0$, it suffices to define an (n, m) matrix X by using only the first m columns of Z, and we have $A = XX'$. This matrix X, a set of n row vectors in $\mathbf{R}^m$, provides the solution for the classical metric scaling problem. As is easily verified by using (4), one finds for the squared Euclidean distances of the points in X the equality

$$\begin{aligned} d^2(x_i, x_j) &= (x_i - x_j)'(x_i - x_j) \\ &= x_i'x_i - x_j'x_i - x_i'x_i + x_j'x_j \\ &= a_{ii} - a_{ij} - a_{ji} + a_{jj} \;=\; d_{ij}^2 \end{aligned}$$

To summarize this result: If the matrix A, based on the given proximity matrix $D = (d_{ij})$, is positive semi-definite with $\text{rank}(A) = m$, we can find n points in $\mathbf{R}^m$ so that their Euclidean distances equal the dissimilarities p_{ij}.

If A is not positive semi-definite. In general, beginning with an arbitrary proximity (dissimilarity) matrix $D = (d_{ij})$, there is no guarantee that the corresponding matrix A is positive semi-definite. The method, discussed above, to find a solution for the classical metric scaling problem is then not applicable. One can try, however, to find an approximative solution. Two approaches have been discussed in the literature.

First, one can nevertheless apply the method described above.[2] Assuming that D is symmetric, all eigenvalues will be real, but not necessarily non-negative. Then, in constructing the configuration X, one can simply use only eigenvectors corresponding to positive eigenvalues. If the

[2] See, for instance, Everitt and Dunn [1991, p. 72].

spectrum of A is dominated by positive eigenvalues, X will provide an approximative solution.

Second, one can try to find a proximity matrix $\tilde{D}$ that comes close to the original matrix D in such a way, that $\tilde{A}$ (constructed by definition (3) applied to $\tilde{D}$) is positive semi-definite. The method to solve the classical scaling problem can then be applied to $\tilde{D}$ instead of D.

In describing this approach, we follow Falk et al. [1995, p. 272]. We assume that A, based on D, is not positive semi-definite. If A's eigenvalues are in descending order, this implies that $\lambda_n < 0$. We can then define a matrix $D_\alpha = (d_{\alpha,ij})$ by

$$d_{\alpha,ij} = d_{ij} + \alpha\,(1 - \delta_{ij})$$

where $\alpha \geq 0$ is a real number and δ_{ij} denotes the Kronecker delta.[3] We want to find an α so that A_α, based on D_α, is positive semi-definite. As shown by Falk et al. [1995, p. 272], this can be achieved with the help of a matrix

$$B = -\frac{1}{2}\,EDE \quad \text{with} \quad E = \left(\delta_{ij} - \frac{1}{n}\right)$$

Let μ_n denote the smallest eigenvalue of B. Then A_α is positive semi-definite for all

$$\alpha \geq \sqrt{4\mu_n^2 - 2\lambda_n} - 2\mu_n$$

We get the smallest α by using the right-hand side of this inequality. One should note, however, that this is not necessarily the smallest α that makes A_α positive semi-definite.

Normalizing configurations. Euclidean distances of the points in a configuration X are invariant with respect to translations and orthogonal rotations. Let X be an (n, m) matrix. We can then add an arbitrary m-vector, say a, to each of its rows, so that

$$d(x_i + a, x_j + a) = d(x_i, x_j)$$

where $d(.,.)$ is used to denote Euclidean distances. Also, if V is an (m, m) orthogonal matrix ($VV' = V'V = I$), then

$$d(Vx_i, Vx_j) = d(x_i, x_j)$$

[3] $\delta_{ij} = 1$ if $i = j$ and zero otherwise.

Therefore, since classical metric scaling is based on Euclidean distances, any orthogonal rotation of the resulting configuration provides a solution. This allows to apply Procrustes rotation to compare different configurations, see Section 7.4.4.1. On the other hand, there is no natural way to select one specific orthogonal rotation. It makes sense, however, to normalize the resulting configuration in such a way that its centroid, defined as

$$\bar{x} = \left(\frac{1}{n}\Sigma_i x_{i1}, \ldots, \frac{1}{n}\Sigma_i x_{im} \right)'$$

becomes zero.

Approximative Solutions. If A, based on a given proximity matrix D, is positive semi-definite, one can find a configuration X in such a way that the Euclidean distances between its row vectors exactly reproduce the given dissimilarities d_{ij}. This requires the rows of X to be in $\mathbf{R}^m$, with $m = \text{rank}(A)$. Of course, to get an approximative solution, one can use less than m, say q, dimensions, corresponding to the q largest eigenvalues of A. The question then is how to measure the quality of the approximation.

An often used measure is based on the relative size of the eigenvalues, see, e.g., Everitt and Dunn [1995, p. 72]. Assuming the eigenvalues are in descending order, the definition is

$$M_q = \sum_{k=1}^{q} \lambda_k \Big/ \sum_{k=1}^{m} \lambda_k$$

With a small modification, this measure can also be applied if A is not positive semi-definite. The definition then becomes

$$M'_q = \sum_{k=1}^{q} \lambda_k \Big/ \sum_{k=1}^{n} |\lambda_k|$$

Of course, these measures only provide some mean value for the quality of approximation. An alternative criterion can be defined as in general metric scaling. The objective function would then be

$$T(X) = \left(\frac{\sum_{j<i} (d(x_i, x_j) - p_{ij})^2}{\sum_{j<i} d(x_i, x_j)^2} \right)^{1/2}$$

Note, however, that using less than m coordinates (m = number of positive eigenvalues of A) does not necessarily minimize $T(X)$. In most cases, direct minimization of $T(X)$ will result in slightly better approximations.

We finally mention that one can look at the problem in a somewhat different way. An accurate solution would be provided by an (n, m) configuration X with $m = \text{rank}(A)$. For a two-dimensional plot, one would only use the first two dimensions, say $\tilde{X}$, corresponding to the two largest eigenvalues of A. Each row of X, say $x_i \in \mathbf{R}^m$, is then projected onto the corresponding row of $\tilde{X}$, that is $\tilde{x}_i \in \mathbf{R}^2$. We can then compare the length of x_i with the length of $\tilde{x}_i$. The indices

$$m_i = \| \tilde{x}_i \| / \| x_i \|$$

provide a measure for the quality of projecting x_i.

7.4.4 Comparing Configurations

This section is intended to discuss methods for comparing configurations. Subsections are as follows.

7.4.4.1 Procrustes Rotation describes a command for Procrustes rotation of configurations.

7.4.4.1 Procrustes Rotation

Assume two (n, m) matrices, X and Y. Procrustes rotation (for an introduction see, e.g., Mardia et al. [1979, pp. 416–419]) tries to find a transformation of Y,

$$\tilde{Y} = \alpha Y R + B \tag{5}$$

in such a way that $\tilde{Y}$ approximates X in a best possible way. Allowed transformations are multiplication with a scalar α, rotation with an (m, m) rotation matrix R, and translation with an (n, m) translation matrix B having identical rows, say b (called a translation vector). In order to measure the approximation one most often uses a least-squares criterion, meaning that one tries to minimize

$$\| X - \tilde{Y} \|_F = \left(\sum_{i=1}^{n} \| x_i - \tilde{y}_i \|^2 \right)^{1/2} \tag{6}$$

subject to the possible transformations defined in (5). Here x_i and $\tilde{y}_i$ denote the rows of X and $\tilde{Y}$, respectively. Assuming a Euclidean norm on the right-hand side, the matrix norm on the left-hand side is called *Frobenius norm*, then

$$\| X - \tilde{Y} \|_F = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} (x_{ij} - \tilde{y}_{ij})^2 \right)^{1/2}$$

It can be shown (for a proof, see Mardia et al. [1979, pp. 416–418]), that minimizing (6) subject to all possible transformations (5) can be achieved with the help of a singular value decomposition of $Z = Y'X$. Let $Z = U\Lambda V'$ be a singular value decomposition of Z, providing two orthogonal (m, m) matrices U and V, and the diagonal matrix Λ containing the singular values of Z. The optimal transformation minimizing (6) is then given by

$$\hat{R} = UV', \quad \hat{b} = \bar{x} - \bar{y}\hat{R}, \quad \hat{\alpha} = \frac{\text{trace}\,(\Lambda)}{\text{trace}\,(YY')} \tag{7}$$

Here $\bar{x}$ and $\bar{y}$ denote row vectors of length m containing the mean values of the columns of X and Y, respectively.

Procrustes rotation in TDA is implemented as a matrix command, `mproc`. The syntax is

```
mproc(X,Y,Y1);
```

where X, Y and $Y1$ are matrix names. X and Y must refer to already defined (n, m) matrices and must have identical dimensions. The command returns the optimally transformed (n, m) matrix $\tilde{Y}$ in `Y1`. If `silent=-1`, the scaling factor α, the translation vector b, and the rotation matrix R will be written into the standard output.

7.4.5 Tree Representations

This section considers some possibilities to use trees for a representation of proximities between a set of objects. Our basic reference is Barthélemy and Guénoche [1991], for an introduction see also Corter [1996]. For our terminology regarding trees see Section **3.6.6**. Here we add some additional terminology and notation.

Tree Distances. In order to give a clear meaning to the problem of representing a proximity matrix by a tree, one needs the concept of tree distance. Let $\mathcal{T}_v = (\mathcal{N}, \mathcal{L}, v)$ denote a valued tree with $\mathcal{N} = \{1, \ldots, N\}$ the set of its nodes, $\mathcal{L}$ the set of edges, and $v(l)$ providing nonnegative values for the edges $l \in \mathcal{L}$. By definition, there is a unique path connecting each pair of nodes $i, j \in \mathcal{N}$. We can then define the *length* of a path as the sum of the values of the edges in the path, and arrive at the following definition of a proximity index for the objects in $\mathcal{N}$:

$$d_v(i,j) \;=\; \text{length of the path connecting } i \text{ and } j$$

Adding the convention that $d_v(i,i) = 0$, this index is a distance as defined in Section **7.4** and will be called a *tree distance* (based on the tree $\mathcal{T}_v$); $(\mathcal{N}, d_v)$ will be called the metric space induced by the tree $\mathcal{T}_v$.

Using this notion of tree distance, a preliminary definition of the tree representation problem can be given as follows: Given a proximity matrix $D = (d(\omega_i, \omega_j))$ for a set of objects, $\Omega = \{\omega_1, \ldots, \omega_N\}$, find a valued tree such that its tree distance, $d_v(i,j)$, in some optimal sense approximates $d(\omega_i, \omega_j)$. As a commonly used measure for evaluating the approximation, we consider the least squares criterion

$$\sum_{\omega_i, \omega_j \in \Omega} \left(d(\omega_i, \omega_j) - d_v(i,j)\right)^2$$

This definition of the tree representation problem is, however, not very general. In fact, we have implicitly assumed a trivial mapping of the objects in Ω to the node set $\mathcal{N}$ and, for many applications, this is too restrictive. A more general formulation can be achieved with the concept of *evolutionary trees* (called 'X-trees' by Barthélemy and Guénoche [1991]).

Evolutionary Trees. In introducing this concept we follow Barthélemy and Guénoche [1991]. An evolutionary tree (wrt Ω) is defined as $\mathcal{T} = (\mathcal{N}, \mathcal{L}, g)$ such that:

1. $(\mathcal{N}, \mathcal{L})$ is a tree,
2. g is a mapping from Ω into $\mathcal{N}$,
3. all nodes in $\mathcal{N} - g(\Omega)$ have degree ≥ 3.

The mapping g is sometimes called the *labeling* of the evolutionary tree; the nodes in $g(\Omega)$ are then called *real nodes* and the nodes in $\mathcal{N} - g(\Omega)$ are called *latent nodes*. In addition we use the following terminology.

1. An evolutionary tree is called *free* if its labeled nodes, $g(\Omega)$ are identical with its leaves, that is, nodes having degree 1.
2. An evolutionary tree is called *constrained* if it does not contain any latent nodes, i.e. $g(\Omega) = \mathcal{N}$.
3. An evolutionary tree is called *separated* if the mapping g is injective, that is, $g(\omega_i) \neq g(\omega_j)$ if $\omega_i \neq \omega_j$.

Like any other tree, an evolutionary tree can be valued by adding a mapping v from the set of its edges, $\mathcal{L}$, into the nonnegative real numbers; we will then use the notation $\mathcal{T} = (\mathcal{N}, \mathcal{L}, g, v)$. Furthermore, one can consider *rooted* evolutionary trees where one of its nodes is selected as a root for the tree.

It is now easily seen that the concept of evolutionary tree provides an essentially more general approach to the tree representation problem. A simple approach only considers constrained evolutionary trees by identifying the objects in Ω with the nodes of the tree. In contrast, a general evolutionary tree may contain any number of latent nodes in order to reach a better representation of the proximities given for Ω.

A more general formulation of the tree representation problem, based on the concept of evolutionary trees, can now be given as follows: Given a proximity matrix $D = (d(\omega_i, \omega_j))$ for a set of objects Ω, find a valued evolutionary tree (wrt Ω) such that its tree distance $d_v(i, j)$ provides a good approximation for $d(\omega_i, \omega_j)$. As before, we take the least squares criterion as our standard measure to evaluate the approximation.

There is obviously a broad range of possibilities for fitting evolutionary trees to a given proximity matrix. The current discussion mainly

focuses on three different kinds of tree distances: centroid, ultrametric, and general additive distances.

Hierarchical Trees. Any rooted tree can be viewed as representing a hierarchy where the place of each node in the hierarchy is defined by the path connecting the node with the root. Using the concept of evolutionary trees, a hierarchical evolutionary tree (wrt Ω) can be defined as a rooted evolutionary tree which is both free and separated. The objects in Ω are then mapped onto the leaves of the tree, and the latent roots provide the hierarchy. While in general the definition of an evolutionary tree requires that all latent roots have degree ≥ 3, we make an exception for the root of a hierarchical evolutionary tree; the root may have a degree ≥ 2.[3]

A hierarchical evolutionary tree can be viewed as a hierarchical clustering scheme (Johnson [1967]) for the set of objects, Ω. To provide a formal definition, let $\mathcal{C}$ denote a set of subsets of Ω. $\mathcal{C}$ is then called a *hierarchical clustering scheme* (HCS) if

1. $\Omega \in \mathcal{C},\ \emptyset \notin \mathcal{C}$
2. for all $\omega \in \Omega$: $\{\omega\} \in \mathcal{C}$
3. for all $C, C' \in \mathcal{C}$: $C \cap C' \in \{C, C', \emptyset\}$.

Defined this way, $\mathcal{C}$ provides a hierarchical clustering of the objects in Ω. At the lowest level, each ω_i constitutes a separate class; the objects are then successively merged into new classes until there is finally only a single class, Ω.

A hierarchical clustering scheme is isomorphic to a hierarchical evolutionary tree (wrt Ω). To see this more clearly, remember the definition of a partial order relation given for general trees in Section 3.6.6. To repeat the definition in the present context, we say that $i \preceq j$, for two nodes $i, j \in \mathcal{N}$, if j lies on the path connecting i with the root of the tree. This in turn leads to the definition of *classes* associated with the nodes of a hierarchical evolutionary tree as follows:

$$C(i) = \{\omega \in \Omega \mid g(\omega) \leq i\} \qquad \text{for} \quad i \in \mathcal{N}$$

The set of classes $C(i)$ then constitutes a hierarchical clustering scheme.

[3] See Barthélemy and Guénoche [1991, p. 22].

7.4.5.1 Hierarchical Trees

To get a better understanding of a hierarchical clustering scheme (HCS), some additional concepts are helpful. As a first step, we note that a HCS can be represented as a rooted tree where the leaves of the tree are identified with the single-object clusters $\{\omega_1\}, \ldots, \{\omega_n\}$, and the root is identified with the single cluster Ω. All other classes of the HCS correspond to internal nodes of the tree. The tree is binary, that is, except for the leaves, all internal nodes of the tree have degree 3, and the root has degree 2.

To see more precisely that a HCS is isomorphic to a rooted tree, let $\mathcal{T} = (\mathcal{N}, \mathcal{L}, r)$ denote a tree with node set $\mathcal{N}$, edge set $\mathcal{L}$, and r the root. We assume that the tree has n leaves representing the objects in Ω. There is then a mapping

$$g : \Omega \longrightarrow \mathcal{N}$$

such that for each $\omega \in \Omega$, $g(\omega)$ is a leave of the tree $\mathcal{T}$.

One can then define a partial order relation as follows: We say that $i \leq j$, for two nodes $i, j \in \mathcal{N}$, if j lies on the path connecting i with the root of the tree. This in turn leads to the definition of classes associated with the nodes of a rooted tree as follows:

$$C(i) = \{\omega \in \Omega \mid g(\omega) \leq i\} \qquad \text{for} \quad i \in \mathcal{N}$$

The set of classes $C(i)$ then constitutes a HCS for the objects in Ω.

To provide additional information about an HCS, we can view the HCS as a sequence of partitions P_r $(r = 0, 1, \ldots, n-1)$. We can then associate with each partition a numerical level providing information about how the partition was created. To define these levels, say $L(P_r)$, we use the dissimilarity measure ρ introduced in the previous section. For the initial partition we set $L(P_0) = 0$. Then, for $r > 0$, we define

$$L(P_r) = \min\{\rho(C, C') \mid C, C' \in P_{r-1}\}$$

Having these numerical levels for the partitions, we can immediately also define a level function for the classes $C \in \mathcal{C}$ by

$$\sigma(C) = \min\{L(P_r) \mid C \in P_r\}$$

Whether we will find a monotonic sequence, $L(P_0) < L(P_1) < \cdots < L(P_{n-1})$ will, in general, depend on the algorithm used to create the HCS. For instance, while the single-link method always generates monotonic sequences of levels, the centroid methods can lead to non-monotonic sequences.[1]

If we find a monotonic sequence of levels, $L(P_r)$, then also the associated level function σ for the classes $C \in \mathcal{C}$ is monotonic in the following sense:

$$\sigma(C) < \sigma(C') \quad \text{if } C \subset C' \text{ and } C \neq C' \tag{8}$$

We can then represent the HCS by a *dendrogram*. This is a special type of a tree representation for $\mathcal{C}$, where the nodes representing the classes $C \in \mathcal{C}$ are plotted according to $\sigma(C)$, that is, according to their distance from the initial partition.

Following Barthélemy and Guénoche (1991, p. 93), we will call an HCS an *indexed hierarchy* if its associated level function is monotonic. There is obviously a close connection with the concept of a dendrogram: an HCS can be represented by a dendrogram if, and only if it can be viewed as an indexed hierarchy.

A further step in investigating the concept of HCS begins with the question how the dissimilarity matrix D is represented by a HCS. In most cases, the HCS provides only a more or less crude "'approximation"'. What is meant by "'approximation"' will be defined below. But we can already note that using a binary tree to represent the dissimilarity matrix for a set of n objects provides only limited degrees of freedom. The number of nodes is $2n-1$, and the number of edges is $2n-2$. While the dissimilarity matrix D can contain up to $n(n-1)/2$ different entries, we can represent only $2n-2$ of these values.

Additional insights are possible if we construct a *valued* tree for representing the HCS $\mathcal{C}$, say $\mathcal{T} = (\mathcal{N}, \mathcal{L}, v, r)$ where $v(l)$ provides the values for $l \in \mathcal{L}$. Each node of the tree represents a class $C \in \mathcal{C}$; the leaves represent the single-object classes, and the root represents Ω. We can then use the level function $\sigma(C)$ to find the value function $v(l)$. This is possible if, and only if the level function is monotonic as defined in (8). Let $l \in \mathcal{L}$ be any edge of the tree. It connects two nodes of the tree, say $i, j \in \mathcal{N}$, representing the classes C_i and C_j of the HCS. We can assume that j lies on a path from i to the root of the tree and, consequently,

[1] See the examples given by Jain and Dubes 1988, p. 84.

$C_i \subset C_j$ and $C_i \neq C_j$. We can then define

$$v(l) = \sigma(C_j) - \sigma(C_i) \tag{9}$$

The monotonicity condition defined in (8) will guarantee that the edge values are positive; and so we arrive at a valued tree that can graphically represented by a dendrogram.

Now remember the concept of tree distance defined in Section **??**. The tree distance $\delta(i, j)$ between every two nodes $i, j \in \mathcal{N}$ is simply the length of the unique path connecting i and j in the tree $\mathcal{T}$ and constitutes a metric distance measure. This tree distance immediately also provides a distance function for Ω by simply restricting δ on the set of leaves representing Ω, or in terms of the mapping g:

$$\delta_c(\omega_i, \omega_j) = \delta_c(g(\omega_i), g(\omega_j))$$

So we arrive at another distance measure for the object set Ω, called the *cophenetic distance measure* on Ω, associated with a HCS. Accordingly, $D_c = (\delta_c(\omega_i, \omega_j))$ is called the *cophenetic matrix* associated with a HCS.

This definition of cophenetic distance implies the monotonicity condition (8). A slightly more general definition refers directly to the levels $L(P_r)$.[2] The definition then becomes

$$\delta_c(\omega_i, \omega_j) = \min\{L(P_r) \mid \omega_i, \omega_j \in C \text{ and } C \in P_r\}$$

Note, however, that in order to represent a cophenetic dissimilarity by a valued tree, the monotonicity condition (8) is crucial.

If we assume this monotonicity, cophenetic dissimilarities become ultrametric distances. In general, a distance or dissimilarity d defined on an object set Ω is called *ultrametric* if the ultrametric inequality holds, that is

$$d(\omega_i, \omega_j) \leq \max\{d(\omega_i, \omega_k) d(\omega_j, \omega_k)\} \quad \text{for all } \omega_i, \omega_j, \omega_k \in \Omega$$

This ultrametric inequality implies the standard triangle inequality and, consequently, an ultrametric dissimilarity index is also a metric distance. Furthermore, since an ultrametric distance for n objects can be completely represented by a valued binary tree with $2n-2$ edges, the distance matrix has at most $2n-2$ distinct values.

For a proof that a cophenetic distance δ_c is ultrametric if, and only if the level function of the HCS is monotonic, we refer to Barthélemy and

[2] See Jain and Dubes 1988, pp. 66–67.

Guénoche (1991, pp. 91–95). They begin with a definition of spherical tree distances. Let δ denote the tree distance constituted by the valued rooted tree $\mathcal{T}$ with root r. δ is called a *spherical* tree distance if $\delta(i,r) = \delta(j,r)$ for all leaves i and J of the tree. The meaning is simply that all leaves have the same distance to the root of the tree. It can be proved, then, that a tree distance δ is spherical if, and only if the induced distance for the leaves of the tree is ultrametric.

Now, assuming the monotonicity condition (8), the tree distance δ defined by (9) is cleary spherical since all paths from the leaves to the root of the tree have the same length. Using the just mentioned result from Barthélemy and Guénoche, it follows that the corresponding cophenetic distance is ultrametric.

On the other hand, assume that we have a representation of the hierarchical clustering scheme $\mathcal{C}$ by a valued tree which induces an ultrametric distance, say δ_c, for the object set Ω. We can then construct an indexed hierarchy. by adding a function σ to $\mathcal{C}$.

Let C be any class in $\mathcal{C}$ represented by a corresponding node k of the tree. If k is a leave of the tree, we simply set $\sigma(C) = 0$. If k is not a leave of the tree, we proceed as follows. We select two leaves of the tree, say i and j such that C is the smallest class containing both ω_i and ω_j. We then have

$$\begin{aligned} \delta(i,r) &= \delta(i,k) + \delta(k,r) \\ \delta(j,r) &= \delta(j,k) + \delta(k,r) \end{aligned}$$

Now, by using the just mentioned result about the equivalence of spherical and ultrametric distances, since δ_c is ultrametric, the corresponding tree distance δ is spherical. It follows by definition that $\delta(i,r) = \delta(j,r)$ and consequently $\delta(i,k) = \delta(j,k)$. So we define $\sigma(C) = \delta(i,k)$. Since this definition is independent of which leaves have been selected, it provides an indexed hierarchy.

So we finally arrive at four equivalent concepts that can be used to characterize an HCS with a monotonic level function:

1. The concept of an indexed hierarchy, $\mathcal{C}_\sigma$, providing a level function σ for the formation of clusters;

2. a dendrogram, providing a graphical display of an indexed hierarchy;

3. a spherical tree distance, δ, for the tree representing the HCS;

4. and a corresponding ultrametric distance, δ_c, for the object set Ω.

7.5 Clustering Procedures

This section deals with clustering procedures based on dissimilarity indices for a given set of objects. We do not consider procedures that implicitly calculate proximity measures from standard (case by variable) data matrices but require that proximities have been established in a separate step. All procedures require as input a valued, and most often also undirected, graph.

7.5.1 Hierarchical Agglomerative Clustering describes hierarchical clustering procedures that follow an agglomerative strategy.

7.5.2 Hierarchical Divisive Clustering describes hierarchical clustering procedures that follow a divisive strategy.

7.5.3 Non-hierarchical Clustering describes non-hierarchical clustering procedures.

7.5.1 Hierarchical Agglomerative Clustering

This section describes procedures for hierarchical agglomerative clustering based on an undirected valued graph where edge values can be interpreted as dissimilarity indices.

7.5.1.1 SAHN Algorithms describes some standard SAHN algorithm for hierarchical agglomerative clustering.

7.5.1.2 Nearest-Neighbor Clustering describes two clustering procedures based on some version of a nearest-neighbor principle.

7.5.1.1 SAHN Algorithms

A hierarchical clustering scheme (HCS) can be viewed as a sequence of partitions $(P_0, P_1, \ldots, P_{n-1})$ of the object set Ω. The initial partition is $P_0 = \{\{\omega_1\}, \ldots, \{\omega\}\}$ containing n single-object classes. Then, at each new level, two classes from the previous partition are merged into a single class in the current partition. This motivates a general algorithm for agglomerative hierarchical clustering as follows:

1. Begin with the partition $P_0 = \{\ \{\omega_i\}, \ldots, \{\omega_n\}\}$ of the object set Ω.
2. Given a partition $P_i = \{C_{i1}, \ldots, C_{in_i}\}$ containing $n_i = n - i$ classes, find two classes C_{ij} and C_{ik} to be merged into a single new class.
3. Create a new partition P_{i+1} by substituting the classes C_{ij} and C_{ik} by the single class $C_{ij} \cup C_{ik}$.
4. As long as the new partition does not equal $\{\Omega\}$ continue with step 2.

The question is how to perform the second step. The most popular family of algorithms is called SAHN (sequential, agglomerative, hierarchical, non-overlapping) algorithms, see Jain and Dubes [1988, p. 79]. The idea is to introduce a dissimilarity measure for clusters and then to select those two clusters for merging that are most similar (least dissimilar) with respect to this measure.

In formal notation, let $P = \{C_1, \ldots, C_m\}$ denote a partition of Ω and $\rho(C_j, C_k)$ a dissimilarity index for the clusters. We have to find two clusters $C_{j'}$ and $C_{k'}$ such that

$$\rho(C_{j'}, C_{k'}) = \min_{j,k} \{\rho(C_j, C_k)\}$$

All SAHN methods use a recursive definition of ρ beginning with

$$\rho(\{\omega_j\}, \{\omega_k\}) = d(\omega_j, \omega_k)$$

Then, in each step of the agglomerative procedure, we have ρ for the current partition which can be used to find two most similar classes for

merging, say C_j and C_k. It then suffices to define ρ for the new partition created by merging the two classes, that is, we need a definition for

$$\rho(C_i, C_j \cup C_k)$$

We shortly describe a few different methods, each based on a different choice of updating ρ. The methods are named according to their description in Jain and Dubes [1988, p. 80]; see also Anderberg [1973, ch. 6], Späth [1975, pp. 170–172]. In describing these methods, $|C_j|$ is used to denote the number of objects in cluster C_j.

Single-Link Method.

$$\rho(C_i, C_j \cup C_k) = \min\{\rho(C_i, C_j), \rho(C_i, C_k)\}$$

Complete-Link Method.

$$\rho(C_i, C_j \cup C_k) = \max\{\rho(C_i, C_j), \rho(C_i, C_k)\}$$

Weighted Average Method.

$$\rho(C_i, C_j \cup C_k) = \frac{1}{2}(\rho(C_i, C_j), \rho(C_i, C_k))$$

Weighted Centroid Method.

$$\rho(C_i, C_j \cup C_k) = \frac{1}{2}(\rho(C_i, C_j), \rho(C_i, C_k)) - \frac{1}{4}\rho(C_j, C_k)$$

Group Average Method.

$$\rho(C_i, C_j \cup C_k) = \frac{|C_j|\rho(C_i, C_j) + |C_k|\rho(C_i, C_k)}{|C_j| + |C_k|}$$

Unweighted Centroid Method.

$$\rho(C_i, C_j \cup C_k) = \frac{|C_j|\rho(C_i, C_j) + |C_k|\rho(C_i, C_k)}{|C_j| + |C_k|} - \frac{|C_j||C_k|}{(|C_j| + |C_k|)^2}\rho(C_j, C_k)$$

Ward's Minimum Variance Method.

$$\rho(C_i, C_j \cup C_k) = \frac{(|C_i| + |C_j|)\rho(C_i, C_j) + (|C_i| + |C_k|)\rho(C_i, C_k) - |C_i|\rho(C_j, C_k)}{|C_i| + |C_j| + |C_k|}$$

Ties in Dissimilarities. In general, the different methods to update ρ lead to different hierarchical clustering schemes. It is therefore an important question how to evaluate the different methods for a specific analytical goal. Some discussion of this question can be found in the literature.[1] An additional problem occurs if the dissimilarity matrix D for the objects in Ω contains ties. Since we have decided to merge only two classes in each step of the agglomerative procedure, ties lead to ambiguities in the sequential ordering of partitions. The only method which behaves well in a situation with tied dissimilarities is the single-link method. All other methods can show results that are more or less influenced by ties, and in so far they depend on the implementation of the algorithm. Unfortunately, there is no convincing solution for this problem; see the discussion in Jain and Dubes [1988, p. 76–79].

Goodness-of-Fit. The question remains how to assess the adequacy of a HCS for a dissimilarity matrix D. The meaning of this question depends on the purpose of the HCS. If the purpose is to finally arrive at a certain partition of the object set Ω, it seems impossible to provide a clear definition of 'adequacy'. If, on the other hand, the purpose is to find a simplifying but in some sense adequate representation of D by using a HCS, it might be possible to define some measure.

Assuming that the purpose is representation, in the context of hierarchical clustering it seems sensible to look for a tree representation. As shown above, having a HCS with a monotonic level function, it can be represented by a valued binary tree. This tree provides a cophenetic matrix $D_c = \delta_c(\omega_i, \omega_j))$ that can be interpreted as a representation of D. So we can compare D and D_c with respect to goodness-of-fit. A simple approach could use a least-squares criterion

$$\Delta(D, D_c) = \sum_{i=2}^{n} \sum_{j=1}^{i} \left(d(\omega_i, \omega_j) - \delta_c(\omega_i, \omega_j)\right)^2$$

If the purpose is to represent D this measure can be used to compare different hierarchical clustering schemes. In fact, if D would be ultrametric, it could be perfectly represented by a monotonic (single-link) HCS, and $\Delta(D, D_c)$ would become zero. Of course, this will be almost never the case in practical applications.

However, the reasoning shows what can be done with hierarchical clustering if the purpose is representation of a dissimilarity matrix D.

[1] See Jain and Dubes [1988, ch. 4].

Box 1 Syntax of `hcls` command

```
hcls (
   opt=...,      option, def. 1
                 1 = single-link
                 2 = complete-link
                 3 = weighted average
                 4 = weighted centroid
                 5 = group average
                 6 = unweighted centroid
                 7 = Ward's minimum variance
   gn=...,       graph number, def. 1
   nfmt=...,     integer print format, def. 4
   fmt=...,      print format for values, def. 10.4
   df=...,       output file with dendrogram
   pcf=...,      plot file for dendrogram
) = fname;
```

These methods assume ultrametric distances as a model for D and based on this assumption, they provide an optimal fit. This then leads immediately to the question whether there are more general models and how they can be fitted.

Implementation. The command for hierarchical clustering with SAHN algorithms is `hcls`. The algorithm is adapted from Späth [1975, p. 172] and supports all seven updating options. It requires a complete dissimilarity matrix without missing values and needs storage for the lower triangle of this matrix.[2]

The syntax of the `hcls` command is shown in Box 1. All parameters except for the name of an output file on the right-hand side are optional. This output file will contain information about the hierarchical merging of objects into clusters. Optionally one can request two more output files. Using the `df` parameter will create an output file that shows the dendrogram as a standard edge list. The `pcf` parameter creates another output file that can be used as a TDA command file to create a PostScript plot of the dendrogram.

In order to illustrate the `hcls` command we use an example data file taken from Späth [1975, p. 168]. It consists of the upper triangle

[2]If n is the number of objects, the required memory is $n(n-1)/2$ times 4 bytes (for single precision floating point numbers).

Box 2 Data file `cl1.dat`

```
 8 50 31 12 48 36  2  5 39 10
   38  9 33 37 22  6  4 14 32
      11 55  1 23 46 41 17 52
         44 13 16 19 25 18 42
            54 53 30 28 45  7
               26 47 40 24 51
                  29 35 34 49
                     3 27 15
                       20 21
                          43
```

Box 3 Output file from `hcls` command

```
Idx    Level     CI    CJ
-----------------------
 1     1.0000     3     6
 2     2.0000     1     8
 3     4.0000     2     9
 4     7.0000     5    11
 5     8.0000     1     2
 6    13.0000     3     4
 7    24.0000     3    10
 8    33.0000     1     5
 9    34.0000     3     7
10    55.0000     1     3
```

of a $(11, 11)$ dissimilarity matrix and is shown in 2.[3] The command file `cl1.cf` uses this data set and performs a complete-link hierarchical clustering. The standard output file is shown in Box 3. The first column numbers the levels. At each level, shown in column 2, the clusters labeled `CI` and `CJ` are merged into a new one. The corresponding columns contain representative node numbers and actually consist of all nodes that have been merged with this representative node number on a lower level.

How this merging actually proceeds becomes visible when looking at the dendrogram shown in 1. It has been created with the output from the `pcf` option. Finally, Box 4 shows the output file created by the `df` parameter. It describes the dendrogram by a standard edge list. Leaves correspond to the node numbers in the dissimilarity matrix, say $1, \ldots, n$. Internal nodes are then added with node numbers $n+1, n+2, \ldots$. The

[3]The name of the data file in the example archive is `cl1.dat`.

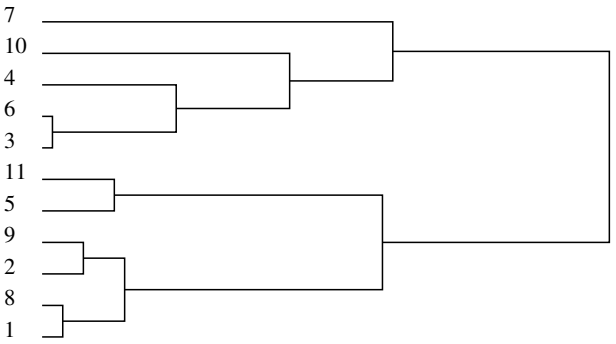


Figure 1 Dendrogram created with `cl1.cf` for a complete-link clustering of the data in Box 2.

Box 4 Output file from `df` option of `hcls` command (`cl1.df`)

I	J	value
3	12	1.00
6	12	1.00
1	13	2.00
8	13	2.00
2	14	4.00
9	14	4.00
5	15	7.00
11	15	7.00
13	16	6.00
14	16	4.00
4	17	13.00
12	17	12.00
10	18	24.00
17	18	11.00
15	19	26.00
16	19	25.00
7	20	34.00
18	20	10.00
19	21	22.00
20	21	21.00

edge values reflect the distances. Notice that this file can be used as an input file to define a new `gdd` data structure. One can use then, for example, the `pltree` command (see section **3.6.6**) to plot different

versions of the dendrogram.[4]

Creating Arbitrary Partitions

The `hcls` command creates a tree (dendrogram) that, in some sense, represents a given dissimilarity matrix, it does not automatically create a partition of the nodes (objects) into clusters. In order to create arbitrary partitions based on the output from the `hcls` command, one can use the `hclsp` command. The syntax is as follows:

```
hclsp (
    nlev=...,          number of levels, def. 0
    cn=...,            list of node numbers
    nfmt=...,          integer print format, def. 4
) = output_file;
```

The command requires a graph defined with the `gdd` command, option 1 (edge list). As an edge list one can use the output file created by the `df` option of the `hcls` command. In any case, the resulting graph must be a tree with a single root node defined by an outdegree 0. There are two options that can be used separately or simultaneously:

a) If nlev > 0, the command determines the root node, that is, the single node with outdegree 0 and reconstructs the tree for `nlev` levels, beginning from the root node. The truncated tree is written to the output file and shows, in addition to the node numbers, the number of leafs that have a directed path to the node. This option might be helpful in selecting a suitable set of internal nodes of the tree to define a partition of the leafs.

b) The second option requires the specification of a sequence of nodes (external node numbers) with the parameter

   `cn =` $n_1, n_2, \ldots,$

   The output file will then contain one record for each leaf of the tree that has a path to one of the nodes $n_1, n_2, \ldots$ Assuming that node i has a path to node n_j, the record will contain two entries: first the node number n_j, followed by the node number i.

[4]Command file `cl1p.cf` provides an example.

Box 5 Command file `cl1a.cf`

```
nvar(
    dfile = cl1.df,
    I = c1,
    J = c2,
    V = c3,
);
gdd = I,J,V;
hclsp(
    nlev=4,
) = cl1a.d;

Content of cl1a.d

(21,11) <- (19,6)  <- (15,2)  <- (5,1)
                              <- (11,1)
                   <- (16,4)  <- (13,2)
                              <- (14,2)
        <- (20,5)  <- (7,1)
                   <- (18,4)  <- (10,1)
                              <- (17,3)
```

Note that it is not required that the nodes $n_1, n_2, \ldots$ define a complete partition of the leafs. However, if a leaf has two or more nodes from the set $n_1, n_2, \ldots$ as followers, the assignment depends on the order of the nodes specified with the `cn` parameter. The algorithm first finds all children of node n_1, then those cildren of node n_2 that were not already found as children of n_1, etc.

To illustrate the command, we use the output from the previous example. The command file is shown in Box 5. The `nvar` command reads the file `cl1.df` shown in Box 4, the `gdd` command creates the graph. By default, this is a directed graph created from the edge list specified by the three variables. Then follows the `hclsp` command with a specification of four levels. The lower part of the box shows the contents of the output file. The first number in each bracketed pair is a node number, the second shows the number of leafs having a path to the node. The output begins with the root node which is 21 in this example and, of course, all 11 leafs belong to this node. The next level is then given by nodes 19 and 20, and so on.

Given this information, one might decide to create two cluster. One cluster that contains all leafs belonging to node 19, and another one

Box 6 Command file `cl1a.cf`

```
nvar(
    dfile = cl1.df,
    I = c1,
    J = c2,
    V = c3,
);
gdd = I,J,V;
hclsp(
    cn = 19, 20,
) = cl1a.d1;

Content of cl1a.d1

  19    5
  19   11
  19    1
  19    8
  19    2
  19    9
  20    7
  20   10
  20    4
  20    3
  20    6
```

containing the leafs belonging to node 20. This can be achieved by using the parameter

```
cn = 19,20,
```

with the `hclsp` command. Box 6 shows the modified command file and, in the lower part of the box, the contents of the output file.

7.5.1.2 Nearest-Neighbor Clustering

This section describes two clustering procedures based on a notion of nearest, or mutual, neighborhood. For a discussion see Jain and Dubes [1988, pp. 128-129]. The command is `nncl` with syntax shown in the following box.

```
nncl (
    alg=...,        algorithm, def. 1
                    1 = nearest neighbors
                    2 = mutual neighborhood
    gn=...,         graph number, def. 1
    sc=...,         threshold (alg=1), def. 1
    ns=...,         size of neighborhood (alg=2), def. 2
    nfmt=...,       integer print format, def. 4
) = fname;
```

All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify an undirected valued graph.

Algorithm 1. In order to describe the algorithm we assume that the graph has n nodes, $i = 1, \ldots, n$; clusters are denoted by $C_1, C_2, \ldots$ The algorithm works as follows:

1. Set $m = 1$, $C_1 = \{1\}$.
2. For $i = 2, \ldots, n$ do the following:
   (a) Find a $j \in C_1 \cup \ldots \cup C_m$ such that $v(i, j)$ is minimal; determine k such that $j \in C_k$.
   (b) If $v(i, j) \leq t$ then $C_k \leftarrow C_k \cup \{i\}$,
   Otherwise $m \leftarrow m + 1$, $C_m = \{i\}$.

The algorithm depends on a parameter, t, that can be selected with the `sc` parameter; by default, $t = 1$. To illustrate the algorithm we use the data introduced in Section 7.5.1.1. The command file is `cl2.cf`; the

Box 1 Output file from `nncl` command, algorithm 1

```
  i  N(i)  CN
--------------
  1    1    1
  2    2    1
  3    3    2
  4    4    1
  5    5    3
  6    6    2
  7    7    4
  8    8    1
  9    9    1
 10   10    5
 11   11    3
```

threshold is `sc = 10`. Box 1 shows the output file. The number of records equals the number of nodes. The first two columns show, respectively, the internal and external node numbers; the third column contains the corresponding cluster number. In this example, with `sc = 10`, we find 5 clusters.

Algorithm 2. The second algorithm is based on mutual neighborhoods having a predefined size, say k, selectable with the `ns` parameter. For each node i, let $n_k(i)$ denote the k nearest neighbors of i. And for each pair of nodes, i and j, define a mutual neighborhood value $m(i, j)$ as follows: if j is the pth nearest neighbor of i, and i is the qth nearest neighbor of j, then $m(i, j) = p + q$; otherwise $m(i, j) = \infty$. The algorithm proceeds as follows:

1. For each node i calculate $n_k(i)$.
2. For each pair of nodes, i and j, calculate $m(i, j)$.
3. For $l = 2, \ldots, 2k$ do the following:
   (a) Consider all pairs of nodes where $m(i, j) = l$, and order these pairs in ascending order w.r.t. their dissimilarity values $v(i, j)$.
   (b) Merge each such pair into a cluster, starting with the pair having the smallest dissimilarity value.

Using the same example data, and `ns = 2`, the output file created by this algorithm is shown in Box 2. The first column numbers the records. The second column shows the level, l, for merging the nodes. The last

Box 2 Output file from `nncl` command, algorithm 2

```
Idx   L    i    j  N(i) N(j)
---------------------------
 1    2    1    8    1    8
 2    2    3    6    3    6
 3    2    5   11    5   11
 4    3    2    9    2    9
 5    3    8    9    8    9
 6    4    3    4    3    4
```

four columns show, respectively, the internal and external node numbers of the nodes that have been merged on the current level.

7.5.2 Hierarchical Divisive Clustering

This section describes procedures for hierarchical divisive clustering based on an undirected valued graph where edge values can be interpreted as dissimilarity indices.

7.5.2.1 Maximally Different Cluster Centers describes a simple divisive clustering scheme that sequentially selects maximally different cluster centers.

7.5.2.2 Partition with Minimal Diameter describes a divisive clustering procedure that sequentially finds clusters with minimal diameter.

7.5.2.1 Maximally Different Cluster Centers

This section describes a very simple divisive clustering scheme that sequentially selects nodes having a maximal dissimilarity. The algorithm goes as follows.

1. Begin with a partition $P_0 = \{C_1\}$ that consists of a single cluster, C_1, containing all nodes of the graph.

2. Given a partition P_m, consider all clusters $C_q \in P_m$.
   If $|\, C_q \,| > n_{\min}$, split C_q into two clusters as follows:

   (a) Find two nodes $i, j \in C_q$ having a maximal dissimilarity.

   (b) Create two sub-classes of C_q as follows:

$$C_{q1} = \{k \in C_q \,|\, v(i,k) \leq v(j,k)\}$$
$$C_{q2} = \{k \in C_q \,|\, v(i,k) > v(j,k)\}$$

3. If no cluster has been split in the previous step, stop. Otherwise create a new partition, P_{m+1}, by substituting all clusters by their sub-classes and continue with step 2.

This simple clustering procedure is implemented as algorithm 1 in the `hcld` command, see Box 1. (The second algorithm will be described in Section 7.5.2.2.) The minimal cluster size, $n_{\min}$, can be selected with the `min` parameter.

To illustrate the output files, we use the example data introduced in Section 7.5.1.1. The command file is `cl3.cf`. The standard output file is shown in Box 2. Each record describes one cluster. The first column shows the level. The second column shows the size of the current cluster. Then follows an enumeration of the nodes that are contained in the current cluster.

A second output file that can be requested with the `df` parameter is shown in Box 3. It contains a graph, given by an edge list, that describes the hierarchical split procedure. The first two columns show node numbers representing the clusters. (Numbers equal those used in the standard output file.) Then follow the respective cluster sizes. The final column shows the distance between the clusters, calculated as one half of the

Box 1 Syntax of `hcld` command

```
hcld(
   alg=...,      algorithm, def. 1
                 1 = max different cluster centers
                 2 = partition with minimal diameter
   gn=...,       graph number, def. 1
   min=...,      minimal cluster size (for alg=1), def. 2
   max=...,      maximal number of splits (for alg=2), def. 1
   df=...,       output file with clustering hierarchy
   nfmt=...,     integer print format, def. 4
   fmt=...,      print format for values, def. 10.4
) = fname;
```

diameters that was used to create the clusters. Since the resulting graph is a tree, it can easily be plotted with the `pltree` command. This is illustrated in Figure 1 that was created with command file `cl3p.cf`.

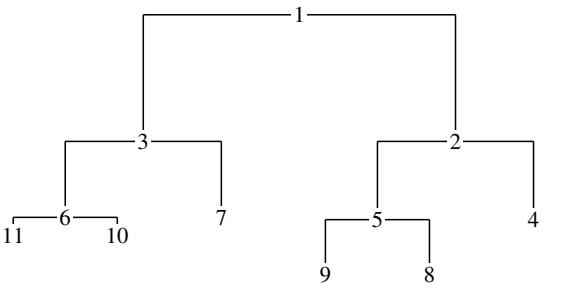


Figure 1 Tree illustrating the divisive clustering procedure, created with command file `cl3p.cf`.

Box 2 Standard output file from `hcld` command, algorithm 1

```
 L    N    nodes
 ------------------------------------------------------------
 1   11    1    2    3    4    5    6    7    8    9   10   11
 2    5    3    4    6    7   10
 3    6    1    2    5    8    9   11
 4    2    4    7
 5    3    3    6   10
 6    4    1    2    8    9
 7    2    5   11
 8    2    3    6
 9    1   10
10    2    1    8
11    2    2    9
```

Box 3 Second output files from `hcld` command, algorithm 1

```
 Ci   Cj   Ni   Nj   distance
 ---------------------------
  2    1    5   11    27.5000
  3    1    6   11    27.5000
  4    2    2    5    17.0000
  5    2    3    5    17.0000
  6    3    4    6    16.5000
  7    3    2    6    16.5000
  8    5    2    3    12.0000
  9    5    1    3    12.0000
 10    6    2    4     4.0000
 11    6    2    4     4.0000
```

7.5.2.2 Partition with Minimal Diameter

This section describes a divisive clustering algorithm that tries to find clusters having minimal diameter. The implementation follows the discussion in Guénoche, Hansen and Jaumard [1991]. In order to describe the algorithm, let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, v)$ denote an undirected valued graph and let the diameter of a cluster $C \in \mathcal{N}$ be defined by

$$d(C) = \max\{v(i,j) \,|\, i,j \in C\}$$

Accordingly, if $P = \{C_1, \ldots, C_m\}$ is a partition of $\mathcal{N}$, its diameter will be defined by

$$d(P) = \max\{d(C) \,|\, C \in P\}$$

Finally, for some set U, let $\mathcal{B}(U)$ denote the set of all partitions of U into two sub-classes. The algorithm proceeds as follows.

1. $m = 1$, $P_1 = \{\mathcal{N}\}$.
2. Select $C_k \in P_m$ such that $d(C_k) = \max\{d(C) \,|\, C \in P_m\}$.
3. Determine a partition $\{C_{2m}, C_{2m+1}\} \in \mathcal{B}(C_k)$ such that $d(\{C_{2m}, C_{2m+1}\}) = \min\{d(\{C_i, C_j\}) \,|\, \{C_i, C_j\} \in \mathcal{B}(C_k)\}$
4. $P_{m+1} \leftarrow (P_m \cup \{C_{2m}, C_{2m+1}\}) - \{C_k\}$
5. $m \leftarrow m + 1$; if $m < m_{\max}$ continue with step 2.

A version of this algorithm, originally proposed by Hubert [1973], is available as algorithm 2 in the `hcld` command described in Section 7.5.2.1. Our implementation follows the discussion of Hubert's algorithm in Guénoche et al. [1991]. The algorithm is based on iteratively calculating, and then bicoloring, maximum spanning trees. This is done with Prim's procedure as described in Section 7.2.5.2. The maximal number of splits, $m_{\max}$, can be selected with the `max` parameter in the `hcld` command.

To illustrate the clustering procedure, we use again the example data introduced in Section 7.5.1.1. The command file is `cl4.cf`. The standard output file is shown in Box 1. Each record describes one cluster. The

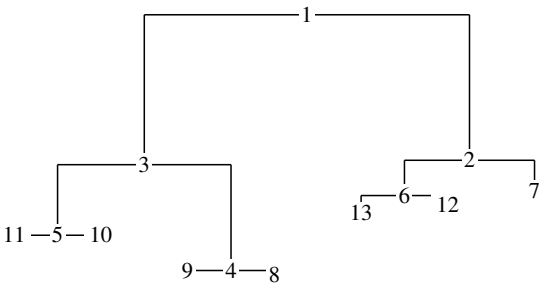


Figure 1 Tree illustrating the divisive clustering procedure, created with command file `cl4p.cf`.

Box 1 Standard output file from `hcld` command, algorithm 2

```
Ci  Cj   diameter    nodes
-----------------------------------------------------
2    1    33.0000    6    1    2    5    8    9   11
3    1    34.0000    5    3    4    6    7   10
4    3    24.0000    3    3    6   10
5    3    16.0000    2    4    7
6    2     8.0000    4    1    2    8    9
7    2     7.0000    2    5   11
8    4     1.0000    2    3    6
9    4     0.0000    1   10
10   5     0.0000    1    4
11   5     0.0000    1    7
12   6     2.0000    2    1    8
13   6     4.0000    2    2    9
```

first column refers to a newly created cluster, the second column refers to its parent cluster. Then follow the diameter of the new cluster and an enumeration of its nodes.

A second output file that can be requested with the `df` parameter is shown in Box 2. It contains a graph, given by an edge list, that describes the hierarchical split procedure. The first two columns show node numbers representing the clusters. (Numbers equal those used in the standard output file.) Then follow the respective cluster sizes and diameters. A plot of the clustering tree, created with the `pltree` command (`cl4p.cf`), is shown in Figure 1.

Box 2 Second output files from `hcld` command, algorithm 2

```
 Ci   Cj   Ni   Nj      d(Ci)       d(Cj)
-------------------------------------------
  2    1    6   11    33.0000     55.0000
  3    1    5   11    34.0000     55.0000
  4    3    3    5    24.0000     34.0000
  5    3    2    5    16.0000     34.0000
  6    2    4    6     8.0000     33.0000
  7    2    2    6     7.0000     33.0000
  8    4    2    3     1.0000     24.0000
  9    4    1    3     0.0000     24.0000
 10    5    1    2     0.0000     16.0000
 11    5    1    2     0.0000     16.0000
 12    6    2    4     2.0000      8.0000
 13    6    2    4     4.0000      8.0000
```

7.5.3 Non-Hierarchical Clustering

This section deals with non-hierarchical clustering procedures. Subsections are as follows.

7.5.3.1 The Bond Energy Approach describes a procedure to re-order the rows and columns of a dissimilarity matrix such that similar nodes are placed in neighboring locations.

7.5.3.1 The Bond Energy Approach

This section describes a clustering approach proposed by McCormick, Schweitzer, and White [1972]; for a more recent discussion see Arabie, Hubert, and Schleutermann [1990]. To describe this approach, let $D = (d_{ij})$ denote a given, not necessarily symmetric, (n, n) proximity matrix, and let π denote a permutation of $\{1, \ldots, n\}$. The idea is to find a permutation of the rows and columns of D such that the following objective function becomes minimal, or maximal.

$$\text{BE}(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{\pi(i),\pi(j)} \left(d_{\pi(i),\pi(j-1)} + d_{\pi(i),\pi(j+1)} + d_{\pi(i-1),\pi(j)} + d_{\pi(i+1),\pi(j)}\right)$$

with the understanding that $d_{\pi(i),\pi(j)} = 0$ if one of the indices is undefined. Since the function is separable into one part for rows and another part for columns, there are three choices. One can optimize only row permutation, only column permutations, or both row and column permutations. If D is symmetric, it will suffice to consider only row, or column, permutations.

As has been pointed out by Lenstra [1974], see also Lenstra et al. [1975], this optimization problem is a special case of the traveling salesman problem and exact solutions can only be obtained for very small number of cases. McCormick et al. [1972] therefore proposed to use some version of a heuristic greedy algorithm to find approximative solutions. In terms of column permutations, the algorithm works as follows:

1. Set $i \leftarrow i_o$, $C = \{i\}$, $n = \{1, \ldots, n\}$

2. For each $j \in N - C$, insert column j of D to the right or the left of the already fixed columns given by C, and calculate the resulting value of the objective function.

3. Having finished the previous step, add that column to C that gave the maximal increase, or decrease, of the objective function.

4. Continue with step 2 until the permuted matrix is complete.

Box 1 Syntax of `becl` command

```
becl (
   alg=...,      algorithm, def. 1
                 1 = only column permutations
                 2 = only row permutations
                 3 = row and column permutations
   min=...,      1 if minimization of objective function
   cn=...,       list of starting nodes, def. 1
   gn=...,       graph number, def. 1
   opt=...,      output option, def. 1
                 1 = permuted node list
                 2 = edge list
                 3 = permuted D, lower triangle
                 4 = permuted D, square matrix
                 5 = like 4, plus leading node numbers
   sc=...,       substitute for missing values, def. -1
                 (only with opt = 3,4,5)
   nfmt=...,     integer print format, def. 4
   fmt=...,      print format for values, def. 10.4
) = fname;
```

This algorithm is provided by the `becl` command. The syntax is shown in Box 1. All parameter, except for the name of an output file on the right-hand side, are optional.

1. By default, the command only performs column permutations. This will suffice for a symmetric proximity matrix. Two other choices can be selected with the `alg` parameter.

2. By default, the command tries to maximize the objective function. The `min=1` parameter can be used to request a minimization.

3. By default, the algorithm uses the starting node $i_0 = 1$. To check possible dependencies, one can provide a list of different starting nodes with the parameter

   `cn =` $i_1, i_2, \ldots,$

   The algorithm is then repeated for each of the specified starting nodes, and the best result is used for output.

To illustrate the output options, we use the example data shown in Box

Box 2 Example data (edge list with five nodes)

```
I J V
-----
3 2 1
4 1 1
4 5 1
5 1 1
```

Box 3 Output files from `becl` command

```
Option 1 (node list)

 i  p(i)   N(i)  p(N(i))
------------------------
 1    3      1       3
 2    2      2       2
 3    5      3       5
 4    4      4       4
 5    1      5       1

Option 2 (edge list)

p(i) p(j)   i    j   N(i) N(j)  value
-------------------------------------
  1    2    3    2     3    2     1
  3    4    5    4     5    4     1
  3    5    5    1     5    1     1
  4    5    4    1     4    1     1

Option 4 (permuted adjacency matrix)
-----------------
0   1   0   0   0
1   0   0   0   0
0   0   0   1   1
0   0   1   0   1
0   0   1   1   0
```

2. It is an edge list for an undirected graph with five nodes. Maximizing the bond energy criterion with column permutations gives the output files shown in Box 3.

7.6 Social Network Analysis

Many of the procedures described in previous sections can also be used to describe and analyze social networks. This chapter is intended to deal with some methods specifically designed for social network analysis; for a general introduction see Wasserman and Faust [1994].

7.6.1 Characterizing Nodes describes procedures that are sometimes useful to characterize the nodes (individuals) in a social network.

7.6.2 Structural Similarity discusses approaches to the question how to define, and find, structural similarities in different embeddings of nodes in a social network.

7.6.1 Characterizing Nodes

Given a social network, an elementary task is to characterize its nodes. While many of the procedures described in earlier chapters can be used for this task, there remain some specific questions which are difficult to approach with those general-purpose commands. This section is therefore intended to describe some more specific commands for the characterization of nodes in social networks.

7.6.1.1 Direct and Indirect Control describes commands to investigate how nodes of a directed graph control other nodes.

7.6.1.2 Integrated Ownership describes the `gio` command that can be used to investigate "integrated ownership".

7.6.1.3 Measures of Flow Control describes a command that allows to investigate how nodes control flows in a directed network.

7.6.1.1 Direct and Indirect Control

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ denote a directed valued graph with adjacency matrix $A = (a_{ij})$. In certain applications we might say that node i directly controls node j if $a_{ij} > s$ where s is some predefined minimum level of influence. Direct control in this sense is easily found with the procedures for direct links discussed in Section 7.2.1.

However, we might also be interested in indirect control. To explain this notion consider the graph in Figure 1 (created with command file `gd12.cf`). Assume that $s = 49$. Node 1 then directly controls only nodes 2 and 3. We might say that this is the first level of control. However, on a second level, node 1 also controls node 5. In general, for each level $l > 1$, if we know the nodes controlled by node 1 for all lower levels, we can also easily find the nodes controlled on the lth level.

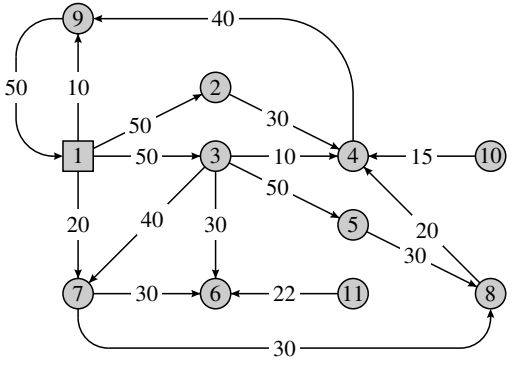


Figure 1 Illustration of direct and indirect control.

Forward Control

TDA offers a command, called `gfcf` (forward control flow), that can be used to find, for any node i, the set of other nodes directly or indirectly controlled by i. The syntax of the command is shown in Box 1. It requires a directed valued graph. Also required is an input file, to be specified with the `if` parameter, containing a list of node numbers. For each node i found in that input file, the `gfcf` command determines the set of nodes

Box 1 Syntax of `gfcf` command (forward control flow)

```
gfcf (
   gn=...,        graph number, def. 1
   if=...,        input file containing node numbers (required)
   sc=...,        minimum level of influence, def. 0.5
   opt=...,       output option, def. 1
                  1 = information for all nodes
                  2 = information only for controlled nodes
   nfmt=...,      integer print format, def. 4
   fmt=...,       print format for values, def. 10.4
) = fname;
```

directly or indirectly controlled by i.

There are two output options depending on the `opt` parameter. If `opt = 1`, the output file will contain information for all nodes in $\mathcal{N}_i$, that is, all nodes which are reachable from i by a directed path, independent of whether a node is controlled by i or not.[1] If `opt = 2`, information will only be given for nodes controlled by i. The contents of the output file will be explained in the following example.

Example 1 For illustration we use the graph shown in Figure 1. (The edge list for this graph is in data file `gd3.dat`.) Having set up a gdd data structure, the command for this illustration is

```
gfcf (sc = 49, if = gd13.if ) = d;
```

The `sc` parameter specifies a minimum level of influence, the `if` parameter provides the name of an input file. In this example, the input file, `gd13.if`, contains just two node numbers, 1 and 9.[2]

The result, in this example the output file `d`, is shown in Box 2. The first two columns contain, respectively, the internal and external node numbers given in the input file, that is, the starting nodes for investigating direct and indirect control. There is one block of records for each of these node numbers. The last two columns show how many records are in each block and provide a counter.

The number of records in each block depends on the `opt` parameter. In our example, we have used the default, `opt = 1`, and each block

[1] There must be, however, at least one node controlled by node i in order to get this information.

[2] The command file is `gd13.cf`.

Box 2 Output from `gfcf` command

```
 i  N(i) N(j)   L      S        M    R    r
---- ----------------------------------------
 1    1    2    1    50.0000    8    8    1
 1    1    3    1    50.0000    8    8    2
 1    1    4    4    60.0000    8    8    3
 1    1    5    2    50.0000    8    8    4
 1    1    6    3    60.0000    8    8    5
 1    1    7    2    60.0000    8    8    6
 1    1    8    3    60.0000    8    8    7
 1    1    9    5    50.0000    8    8    8

 9    9    1    1    50.0000    8    8    1
 9    9    2    2    50.0000    8    8    2
 9    9    3    2    50.0000    8    8    3
 9    9    4    5    60.0000    8    8    4
 9    9    5    3    50.0000    8    8    5
 9    9    6    4    60.0000    8    8    6
 9    9    7    3    60.0000    8    8    7
 9    9    8    4    60.0000    8    8    8
```

contains a separate record for each node that can be reached from the starting node. For example, there are 8 nodes that can be reached from node 1. The fourth column, labeled `L`, shows whether `N(j)` is controlled by `N(i)` and, given that this is the case, the corresponding level of influence. (The entry will be -1 if there is no control.) In our example, we find that node 1 controls nodes 2 and 3 on the first level, nodes 5 and 7 on the second level, nodes 6 and 8 on the third level, node 4 on the fourth level, and node 9 on the fifth level.

Backward Control

As a complementary question, we might want to know for some specified node i by which other nodes it is controlled. This question, of course, makes only sense if i has some positive in-degree, i.e., is not an ultimate node. Answering this question is somewhat more computationally involved compared with the question for forward control discussed above. One firstly needs to find all nodes that possibly control some other node.

Another concern is that the question for backward control is not immediately precise. Do we want to find all nodes that directly or indirectly control node i? This would then require to record a possibly very complicated graph that, furthermore, may also contain cycles. We

Box 3 Syntax of `gbcf` command (backward control flow)

```
gbcf (
   gn=...,        graph number, def. 1
   if=...,        input file containing node numbers (optional)
   sc=...,        minimum level of influence, def. 0.5
   opt=...,       output option, def. 1
   nfmt=...,      integer print format, def. 4
   fmt=...,       print format for values, def. 10.4
   df=...,        optional second output file
   ns=...,        max number of controlled nodes
) = fname;
```

shall, therefore, deal only with the somewhat simpler question: to find all nodes that might control node i but are not controlled by some other node.

The TDA command for this task is `gbcf` (backward control flow) with syntax shown in Box 3. The command requires a directed valued graph. All parameters, except for the name of an output file to be given on the right-hand side, are optional.

In a first step, the command tries to find all nodes with zero in-degree that possibly control some other node. This step requires sufficient memory to keep a list of all these nodes. By default, the maximum number of these nodes is two times the number of nodes in the graph. A different maximum number of nodes can be specified with the `ns` parameter.

If the `df` parameter is used to request a second output file, the list of these nodes is written into that file. To be precise, the file will contain three columns. The first two contain, respectively, the internal and external node numbers, the third column records how many nodes are directly or indirectly controlled by that node. Let $\mathcal{U}$ denote the set of these nodes which directly or indirectly control at least one other node. If this set is empty the command terminates.

In a second step, the command considers, in turn, each input node. By default, these are all nodes in the graph with positive in-degree. Optionally, a list of input nodes may be specified with the `if` parameter. For each input node, say i, the command determines a set, say $\mathcal{N}_i^*$, consisting of all nodes that have a direct forward link to node i.

In a third step, the command checks whether some, or all, of the nodes in $\mathcal{N}_i^*$ are controlled by nodes in $\mathcal{U}$. These nodes are then substituted by the corresponding nodes in $\mathcal{U}$.

Output is written into the file specified on the right-hand side of the command. The contents of this file depend on the `opt` parameter. There are two options (`opt=1`, or `opt=2`) as will be explained in the following example.

Example 2 If the `gbcf` command is used with the graph shown in Figure 1 it will give the message that there are only two nodes with zero in-degree (nodes 10 and 11) and neither of these nodes controls some other node. Consequently, the command terminates at the end of the first step. To provide a somewhat more interesting example we change the graph by substituting the value of the edge $(10, 4)$ from 15 into 55, and deleting the edge $(9, 1)$. The new data file is `gd3a.dat`. We then try two versions of the command. The first one (command file `gd14.cf`) is

```
gbcf (sc = 49, df = df) = da;
```

that uses the default `opt=1` to create the output file `da` and creates the additional output file `df`. A second version (command file `gd14a.cf`) is

```
gbcf (sc = 49, opt = 2) = db;
```

to create another output file, `db`, based on the second output option. All output files are shown in Box 4.

1. The output file `df` shows that the graph contains three nodes with zero in-degree, node number 1, 10, and 11. Node 1 controls 8 other nodes, node 10 controls one other node, and node 11 controls no other node.

2. The output file `da` contains a variable number of records for each node having a positive in-degree. The first two columns record the internal and external node number, i and $N(i)$, respectively. The column labeled `NA` shows the number of records in the block, identical with the in-degree of the node. Then follow the node number that is directly linked to node i and the value of the corresponding edge. For instance, there is one node, 1, directly linked to node 2, with edge value 50; and there are for nodes linked to node 4 with the corresponding edge values as shown in the file.

Then follows a column labeled `NY` that provides the number of "ultimate" nodes which might control node i. For example, node 4 has in-degree 4, but is controlled by only two other nodes, namely 1 and 10, as is recorded in the column labeled `N(k)`. The final column shows the

Box 4 Output files from `gbcf` command

```
output file: df

  i  N(i)   M
-------------
  1    1    8
 10   10    1
 11   11    0

output file: da

 i  N(i)  NA  N(j)    a(j,i)   NY  N(k)    y(k,i)
-------------------------------------------------
 2    2    1    1   50.0000    1    1   50.0000
 3    3    1    1   50.0000    1    1   50.0000
 4    4    4    2   30.0000    2    1   60.0000
 4    4    4    3   10.0000    2   10   55.0000
 4    4    4    8   20.0000    2   -1    0.0000
 4    4    4   10   55.0000    2   -1    0.0000
 5    5    1    3   50.0000    1    1   50.0000
 6    6    3    3   30.0000    2    1   60.0000
 6    6    3    7   30.0000    2   11   22.0000
 6    6    3   11   22.0000    2   -1    0.0000
 7    7    2    1   20.0000    1    1   60.0000
 7    7    2    3   40.0000    1   -1    0.0000
 8    8    2    5   30.0000    1    1   60.0000
 8    8    2    7   30.0000    1   -1    0.0000
 9    9    2    1   10.0000    1    1   50.0000
 9    9    2    4   40.0000    1   -1    0.0000

output file: db

 i  N(i)   L   NA   NY  ------ A -----  ------ Y -----
-------------------------------------------------------
 2    2    4    1    1  50   0   0   0  50   0   0   0
 3    3    4    1    1  50   0   0   0  50   0   0   0
 4    4    4    4    2  55  30  20  10  60  55   0   0
 5    5    4    1    1  50   0   0   0  50   0   0   0
 6    6    4    3    2  30  30  22   0  60  22   0   0
 7    7    4    2    1  40  20   0   0  60   0   0   0
 8    8    4    2    1  30  30   0   0  60   0   0   0
 9    9    4    2    1  40  10   0   0  50   0   0   0
```

aggregated edge values for the "ultimate" nodes that control node i. For example, node 1 (possibly) controls node 4 with an aggregated value of 60 (whatever units). Note that the algorithm does not check whether the edge values are proper fractions.

3. Output file `db`, based on `opt=2`, provides part of the information in a somewhat different form that can easier be used for calculating descriptive statistics. There is only one record for each of the input nodes. The column labeled `L` records the number of entries in the columns of A and Y. It is, in fact, the maximum of the in-degrees for all nodes. The columns labeled `NA` and `NY` show the number of non-zero entries in the corresponding rows of A and Y. Then follow `L` columns with the entries of A, that is, values of the edges directly ending in node i. Finally, there are `L` columns with the entries of Y, i.e., the edge values after aggregation.

7.6.1.2 Integrated Ownership

Consider graph A shown in Figure 1. It is a simple directed and valued graph. Assume that we can associate with each node i some value, or resource, R_i. Denoting the graph's adjacency matrix by $A = (a_{ij})$, we may then interpret a_{ij} as the fraction of R_j directly owned by node i. For example, node 1 directly owns 0.5 of R_2.

Given this situation, we might also assume that ownership is transitive along all directed paths in the graph. It then follows, for example, that node 1 owns (directly and indirectly) $0.5 \cdot 0.3$ of R_4. Similarly, to calculate what node 1 owns of R_3, we need to add $0.5 \cdot 0.3 \cdot 0.2 + 0.5 \cdot 0.1$.

These calculations are straightforward as long as the graph has no cycles. Otherwise one has to take into account how indirect ownership evolves through the cycles. See, for example, graph B in Figure 1. Obviously, there is a cycle: node 2 owns some fraction of R_3, but node 3 also owns some fraction of R_2.

Following a proposal by Baldone, Brioschi, and Paleari [1997], one can use ideas from input-output analysis to find a suitable approach. Let y_{ij} denote the fraction of R_j owned by node i directly or indirectly. We may then write the accounting equation

$$y_{ij} = a_{ij} + \sum_{k \neq i} y_{ik} a_{kj}$$

or, equivalently,

$$y_{ij} = (1 - y_{ii}) a_{ij} + \sum_{k} y_{ik} a_{kj}$$

Using I_n to denote the unit matrix of order n, and $Y = (y_{ij})$, this may also be written in matrix notation as

$$Y = (I_n - \mathrm{diag}(Y))A + YA \tag{10}$$

As shown by Baldone et al. [1997], this equation can be solved in Y. Sufficient conditions are that the graph, and consequently its adjacency matrix, is connected and $0 \leq a_{ij} \leq 1$. First, it directly follows from (10) that

$$(I_n - \mathrm{diag}(Y))^{-1} Y = A(I_n - A)^{-1} \tag{11}$$

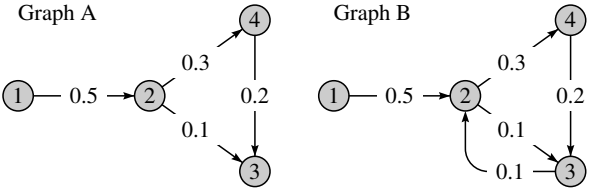


Figure 1 Two simple directed valued graphs. Graph A without, graph B with a cycle (`gd9.cf`).

We may then write

$$\begin{aligned}\text{diag}\{A(I_n - A)^{-1}\} &= \text{diag}\{(I_n - \text{diag}(Y))^{-1}Y\} \\ &= \text{diag}\{(I_n + \text{diag}(Y) + \text{diag}(Y)^2 + \ldots)Y\} \\ &= \text{diag}(Y) + \text{diag}(Y)^2 + \ldots) \\ &= (I_n - \text{diag}(Y))^{-1} - I_n\end{aligned}$$

Consequently,

$$(I_n - \text{diag}(Y))^{-1} = I_n + \text{diag}(A(I_n - A)^{-1})$$

Inserting into (11) gives

$$\begin{aligned}A(I_n - A)^{-1} &= \text{diag}(I_n + A(I_n - A)^{-1})Y \\ &= \text{diag}(I_n + A + A^2 + \ldots)Y \\ &= \text{diag}((I_n - A)^{-1})Y\end{aligned}$$

Finally, we get the result

$$Y = \text{diag}((I_n - A)^{-1})^{-1}A(I_n - A)^{-1} \tag{12}$$

For small and medium-sized graphs this equation can be solved by standard matrix procedures. Example 1 will provide an illustration. For large graphs one needs a somewhat different approach that will be explained below.

Example 1 To illustrate formula (12) we use data for the graphs shown in Figure 1. Edge lists and adjacency matrices for both graphs are shown in Box 1. For calculating the Y matrices from formula (12) we use TDA's

Box 1 Example data files

```
Graph A                          Graph A

edge list (gd2a.dat)             edge list (gd2b.dat)
--------------------             --------------------
      1 2 0.5                          1 2 0.5
      2 3 0.1                          2 3 0.1
      2 4 0.3                          2 4 0.3
      4 3 0.2                          4 3 0.2
                                       3 2 0.1

adj. matrix (gd2a.mat)           adj. matrix (gd2b.mat)
----------------------           ----------------------
   0.0 0.5 0.0 0.0                  0.0 0.5 0.0 0.0
   0.0 0.0 0.1 0.3                  0.0 0.0 0.1 0.3
   0.0 0.0 0.0 0.0                  0.0 0.1 0.0 0.0
   0.0 0.0 0.2 0.0                  0.0 0.0 0.2 0.0
```

Box 2 Example command file `gd10.cf`

```
mfmt= 7.4;
mdef(A,4,4)=gd2a.mat;
mdefi(4,4,I);
mexpr(I - A,IA);
mginv(IA,IAI);
msqrti(IAI,IADI);
mmul(IADI,IADI,A,IAI,Y);
mpr(Y);
```

Box 3 Y matrices calculated with `gd10.cf`

```
          Graph A                              Graph B
------------------------------     ------------------------------
0.0000  0.5000  0.0800  0.1500     0.0000  0.5081  0.0813  0.1524
0.0000  0.0000  0.1600  0.3000     0.0000  0.0160  0.1600  0.3000
0.0000  0.0000  0.0000  0.0000     0.0000  0.1000  0.0160  0.0300
0.0000  0.0000  0.2000 -0.0000     0.0000  0.0202  0.2020  0.0061
```

matrix commands. A command file is shown in Box 2. (This command file uses the adjacency matrix for graph A. To calculate Y for graph B simply change the name of the input file. For other matrices it might be necessary to change also the matrix dimensions.) The resulting Y matrices for both graphs are shown in Box 3.

Box 4 Algorithm for `gio` command

(1) determine $\mathcal{N}_i$

(2) set: $u_i \leftarrow a_i, y_i \leftarrow 0$

(3) $\forall j \in \mathcal{N}_i \; \forall (k,j) \in \mathcal{E}_j$: if $(k = i)$ then $y_{ij} \leftarrow y_{ij} + v(k,j)$
else $y_{ij} \leftarrow y_{ij} + u_{ik} v(k,j)$

(4) if $(\max_j \{ \mid u_{ij} - y_{ij} \mid \} \leq \epsilon)$ then end

otherwise set: $u_i \leftarrow y_i, y_i \leftarrow 0$ and continue with (3)

A Procedure for Large Graphs

Standard matrix procedures are no longer feasible if the graph becomes large. For example, we might be interested in investigating integrated ownership based on share-holding data for a network of firms. The number of firms can easily exceed 30000.[1] The adjacency matrix would then become a 30000×30000 matrix and assuming that we need 8 bytes for each element storage requirements for this matrix would exceed 7 giga bytes. So we need a different approach.

Our approach uses an iterative scheme that can be applied to each node separately. Then, given some node number i, we only need to calculate the ith row of Y, denoted by Y_i. While this can be done for each node in turn, one is often only interested in a subset of nodes, for example, all nodes with zero in-degree (sometimes called "ultimate" nodes). The procedure therefore allows to specify a subset of nodes to be used for calculating the corresponding rows of Y.

Further computational savings are possible due to the fact that the adjacency matrix, A, is most often very sparse. In fact, in order to calculate Y_i, we only need to consider the set of nodes that can be reached from i by a directed path, denoted by $\mathcal{N}_i$, taking into account, of course, all cycles that may be present in the graph. We often find that the number of nodes in $\mathcal{N}_i$ is substantially smaller than in $\mathcal{N}$.

To explain the algorithm let i be a node number, $\mathcal{N}_i$ as defined above, let $\mathcal{E}_j$ denote the set of edges ending in node j (excluding loops) and

[1] We thank Marco Becht who brought our attention to this application.

Box 5 Syntax of `gio` command

```
gio (
   gn=...,        graph number, def. 1
   if=...,        input file containing node numbers
   mxit=...,      maximum number of iterations, def. 20
   eps=...,       epsilon for convergence, def. 0.0001
   opt=...,       output option, def. 1
                  1 = print all y_ij values
                  2 = print only values where a_ij ≥ s or y_ij ≥ s
   sc=...,        s value for opt=2, def. 0.0
   nfmt=...,      integer print format, def. 4
   fmt=...,       print format for values, def. 10.4
) = fname;
```

$v(k, j)$ the value of an edge $(k, j) \in \mathcal{E}_j$. Also let a_i denote the ith row of A, and let u_i and y_i be two row vectors with the same dimension as a_i. How the algorithm works is shown in Box 4. If this algorithm terminates successfully, that is, reaches the required accuracy specified by ϵ within a certain limit of iterations, y_i will contain the required elements of the ith row of Y.

The `gio` command. The algorithm described above is performed by a command called `gio` with syntax shown in Box 5. By default, the command performs the calculations for each node in the graph. Optionally, one can provide an additional input file with the `if` parameter that contains a list of node numbers. y_i is then only calculated for nodes specified in that input file.

There are two output options. If `opt = 1` (default) the output file will contain a separate record for each $j \in \mathcal{N}_i$. If `opt = 2`, it will only contain records where $a_{ij} \geq s$ or $y_{ij} \geq s$ where s is some non-negative value that can be specified with the `sc` parameter. The contents of the output file will be explained in Example 2.

Example 2 To illustrate the `gio` command we continue with Example 1. Since the command needs a gdd data structure we now use the edge list file `gd2a.dat` and `gd2b.dat`, shown in Box 1. Having created a gdd data structure, the command can be used without any parameters as

```
gio = d;
```

Box 6 Output from `gio` command

```
i Ni Nj  A(i,j)  Y(i,j)  C IT
---------------------------
1  1  1  0.0000  0.0000  4  1
1  1  2  0.5000  0.5000  4  1
1  1  3  0.0000  0.0800  4  1
1  1  4  0.0000  0.1500  4  1
2  2  2  0.0000  0.0000  3  1
2  2  3  0.1000  0.1600  3  1
2  2  4  0.3000  0.3000  3  1
3  3  3  0.0000  0.0000  1  1
4  4  3  0.2000  0.2000  2  1
4  4  4  0.0000  0.0000  2  1
```

```
i Ni Nj  A(i,j)  Y(i,j)  C IT
---------------------------
1  1  1  0.0000  0.0000  4  4
1  1  2  0.5000  0.5081  4  4
1  1  3  0.0000  0.0813  4  4
1  1  4  0.0000  0.1524  4  4
2  2  2  0.0000  0.0160  3  2
2  2  3  0.1000  0.1600  3  2
2  2  4  0.3000  0.3000  3  2
3  3  2  0.1000  0.1000  3  1
3  3  3  0.0000  0.0160  3  1
3  3  4  0.0000  0.0300  3  1
4  4  2  0.0000  0.0202  3  2
4  4  3  0.2000  0.2020  3  2
4  4  4  0.0000  0.0061  3  2
```

where `d` is the name of an output file.[2] Box 6 shows shows the resulting output files. They are obviously identical with the output files in Box 3 that were created with standard matrix procedures.

Note that the output files contain two additional columns. The column labeled `C` shows the number of nodes in $\mathcal{N}_i$. The column labeled `IT` shows the number of iterations used to calculated the corresponding value of y_{ij}. If this number equals the maximum number of iterations, the algorithm has most probably not reached the required accuracy specified with the `eps` parameter. Note that for graph A the number of iterations is always 1. Since the graph does not contain cycles the algorithm didn't need to iterate.

[2]The command file is `gd11.cf`.

7.6.1.3 Measures of Flow Control

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, v)$ denote a directed and valued graph. As described in Section 7.2.8.1, we can then consider, for each pair of nodes, $i, j \in \mathcal{N}$, the maximal flow from i to j. This then allows to characterize nodes by their ability to control the possible flows in the network; for a discussion see Freeman et al. [1991]. The algorithm works as follows.

1. Consider all pairs of nodes, $i, j \in \mathcal{N}$, where there is at least one directed path from i to j, and calculate the maximal flow from i to j, denoted by f_{ij}.

2. Then consider all nodes $k \in \mathcal{N}$. If $k = i$, or $k = j$, or k is not part of at least one directed path from i to j, set $f_{ij,k} = -1$. Otherwise delete k and all of its adjacent edges from $\mathcal{G}$ and let $f_{ij,k}$ be the maximal flow from i to j in the reduced graph.

By comparing f_{ij} and $f_{ij,k}$ one can finally assess the capability of node k to control, or not to control, any flows in the network. And, of course, one can think of a lot of measures to summarize these capabilities.

```
gfc (
    gn=...,          graph number, def. 1
    nfmt=...,        integer print format, def. 4
    fmt=...,         print format for flow values, def. 10.4
) = fname;
```

The `gfc` command performs the necessary calculations. All parameters, except for the name of an output file on the right-hand side, are optional. The current relational data structure must specify a directed valued graph that must be defined with option 1 of the `gdd` command, i.e. as an edge list. The algorithm for calculating maximal flows is the same as used for the `gflow` command, see 7.2.8.1.

Example 1 To illustrate the `gfc` command we use an example discussed in Freeman et al. [1991]. The graph is shown in Figure 1. The corresponding data file is `gd14.dat`, the command file for the examples is `gd41.cf`. The resulting output file is shown in Box 1.

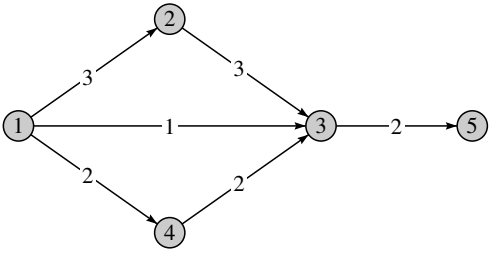


Figure 1 Directed graph used for examples (`gd40.cf`).

Box 1 Output file from `gfc` command

```
 i    j  N(i) N(j) f(ij)  f(ij,1)   ...    f(ij,5)
-------------------------------------------------------
 1    2    1    2    3     -1   -1   -1   -1   -1
 1    3    1    3    6     -1    3   -1    4   -1
 1    4    1    4    2     -1   -1   -1   -1   -1
 1    5    1    5    2     -1    2    0    2   -1
 2    3    2    3    3     -1   -1   -1   -1   -1
 2    5    2    5    2     -1   -1    0   -1   -1
 3    5    3    5    2     -1   -1   -1   -1   -1
 4    3    4    3    2     -1   -1   -1   -1   -1
 4    5    4    5    2     -1   -1    0   -1   -1
```

7.6.2 Structural Similarity

A broad variety of methods can be used to compare nodes in a social network. One approach begins with a characterization of each node separately and then compares the individually assigned measures, e.g., by investigating their frequency distribution or by some summary statistics. A different approach tries to establish, and then represent, proximity relations between the nodes in the network. This can be done in two different ways; for a discussion see Burt [1978]. One approach relies on directly interpreting the relationships given by the network as providing information about proximities between nodes. This then leads to a search for "cohesive subgroups", using whatever kind of clustering procedure one likes and is available. A different approach focuses on how the nodes are embedded in the network. One is interested, then, in a comparison of nodes with respect to the question whether their embedding in the network shows similarities. Of course, it might then be possible to finally arrive at a second-level definition of proximities between nodes, and apply standard methods to represent these proximities, or to apply clustering procedures (sometimes called "block modeling").

The current section is intended to deal with this approach to finding "structural similarities" between nodes, based on an investigation of the way nodes are embedded in a network. Subsections are as follows.

8. Set-Valued Data

This part contains the following sections.

8.1 Introduction and Overview explains our notion of set-valued and, in particular, interval-valued variables.

8.2 Interval Expressions explains how to define and evaluate interval expressions.

8.3 Functions with Interval Arguments explains our notion of functions that are defined by interval expressions and describes a command that can be used to calculate inclusion functions.

8.4 Range of Interval Functions describes a command for global function minimization, based on inclusion functions, and a similar command that finds the range of an interval function.

8.5 Distribution Functions describes commands that can be used to calculate distribution functions for set-valued and interval-valued variables.

8.6 Descriptive Statistics describes commands that calculate elementary descriptive statistics for interval-valued variables.

8.1 Introduction and Overview

A statistical variable can be defined as a mapping

$$X : \Omega \longrightarrow \mathcal{X}$$

Ω refers to a given set of individuals (objects), and $\mathcal{X}$ is a property space. The variable, X, associates with each individual $\omega \in \Omega$ a specific property $X(\omega) \in \mathcal{X}$. In all applications we also assume that we are given a numerical representation for the property space that can formally be identified with $\mathcal{X}$. Most often this will be subset of the natural or real numbers. The essential point is that we normally also assume that $X(\omega)$ can be assumed to be an exactly identified numerical value in $\mathcal{X}$, i.e., the numerical representation of the property space. An interesting alternative comes into view if we only assume that we are given, for each individual $\omega \in \Omega$, only a subset of possible values. This leads to the idea of a *set-valued variable* that can formally be defined as a mapping

$$X : \Omega \longrightarrow \mathcal{P}(\mathcal{X})$$

where $\mathcal{P}(\mathcal{X})$ now refers to the power set of $\mathcal{X}$. This notion also allows a new view on missing values. There is no longer a fundamental distinction between, on the one hand, exactly given values and, on the other hand, completely missing values. Instead, depending on the size of $X(\omega)$, values of set-valued variables are more or less precise. Missing values, in the traditional sense of the word, are then those extreme cases where $X(\omega)$ equals the given property set, $\mathcal{X}$.

This part of the manual deals with set-valued variables. We distinguish two kinds of such variables. First, *discrete variables* which are defined by a discrete property space, i.e., its numerical representation is given by a subset of the natural numbers. Values of discrete set-valued variables are then also given as subsets of discrete values. Second, we shall deal with *interval-valued variables*. In this case the property space has a numerical representation that is given by some interval of real numbers and it is assumed that values of the variable are also given by intervals. We furthermore assume that these intervals have a positive width, that is, we exclude the possibility of exactly given real values.

Command		Section
`evalfi`	calculates inclusion functions	8.3.2
`gmin`	global minimum of a function	8.4.1
`iddf`	distribution function	8.5.1
`idf`	distribution function	8.5.2
`imean`	mean value	8.6.1
`ivar`	variance	8.6.2
`mpr`	evaluation of interval expression	8.2.2
`range`	range of a function	8.4.2
`sddf`	distribution function	8.5.1

8.2 Interval Expressions

This section deals with interval expressions, that is, expressions that may contain interval-valued arguments. There are two subsections.

8.2.1 Definition of Interval Expressions explains the available operators that can be used to specify interval expressions.

8.2.2 Evaluation of Interval Expressions describes an enhancement of the `mpr` command that can be used to evaluate interval expressions.

8.2.1 Definition of Interval Expressions

An *elementary interval expression* has the form

$$\texttt{iv}(\, e_1, e_2 \,)$$

`iv` is the name of an operator, called *interval operator*, an e_1 and e_2 are standard expressions. Alternatively, one can use the syntax

$$[\, e_1, e_2 \,]$$

A (general) *interval expression* is an expression that contains at least one elementary interval expression.

The general idea is that one can begin with elementary interval expressions and then, by applying operators, can create more complex interval expressions. Only a subset of TDA's operators can be used, however, to create interval expressions. They will be described in turn. The basic principle is that the result of applying an operator to an interval should result in another interval that contains all possible values that result from applying the operator to any value of the interval that is given as an argument. Thus, for an operator that has a single argument, its application should result in

$$\mathrm{op}(\, [\, e_1, e_2 \,] \,) \;=\; \{\, \mathrm{op}(e) \,|\, e \in [\, e_1, e_2 \,] \,\}$$

Correspondingly, for an operator with two arguments we have

$$\mathrm{op}(\, [\, e_1, e_2 \,], [\, e_1', e_2' \,] \,) \;=\; \{\, \mathrm{op}(e, e') \,|\, e \in [\, e_1, e_2 \,], e' \in [\, e_1, e_2 \,] \,\}$$

The following operators can be used to create interval expressions. Notice that these are always type 1 operators; type 2 operators cannot be used to create interval expressions.

1. The basic numerical operators: $+, -, *, /$.

2. The operators: sin(), cos(), log(), exp().

3. The operators: min() and max().

Note 1. Contrary to our definition of interval-valued variables, interval expression may contain degenerate intervals, that is, intervals $[e_1, e_2]$

where $e_1 = e_2$. It is also possible to use an elementary interval expression where $e_2 < e_1$. In fact, when evaluating interval expressions, an elementary interval expression given by $[\, e_1, e_2 \,]$ is always taken as

$$[\, \min(e_1, e_2), \max(e_1, e2) \,]$$

Note 2. For a good introduction to interval calculations and some of its applications see Hansen (1992). In the literature, one main reason for using interval calculations is that, in principle, this approach allows to cope with the problem of inaccuracies and rounding errors when performing numerical calculations on a computer. This requires, however, proper rounding procedures (outward rounding of interval bounds). We stress that TDA's interval procedures do not apply proper rounding. In fact, TDA's interval procedures are not intended to correctly deal with problems of numerical inaccuracy and rounding errors. They definitely serve another purpose: to deal with interval-valued variables where intervals are most often very broad and one can neglect problems that may occur in the calculation, and proper rounding, of interval bounds.

8.2.2 Evaluation of Interval Expressions

In order to evaluate interval expressions one can use the `mpr` command. The syntax is

```
mpr(expression [, string] ) [= filename];
```

`expression` can be an interval expression. The result is written as an interval. The lower and upper bounds of the interval are not enclosed by square brackets. In order to get surrounding square brackets one can use `mpr1`, instead of `mpr`.

8.3 Functions with Interval Arguments

This section deals with functions that may contain interval-valued arguments. There are two subsections.

8.3.1 Definition of Interval Functions explains how to specify functions that may contain interval-valued arguments.

8.3.2 Evaluation of Inclusion Functions describes the `evalfi` command that can be used to find inclusion functions for user-defined interval functions and first derivatives.

8.3.1 Definition of Interval Functions

We use the term *interval function* to denote an interval expression that contains one or more free variables. For example,

$$f(x) := \sin(x + [\,1, 2\,])$$

would be an interval function with a single variable (argument), x. The expression that defines an interval function may, or may not, contain elementary interval expressions. In any case, it is required that the expression can be evaluated as an interval expression, that is, the expression must only contain operators that are allowed for interval expressions.

In most applications it is also required that domains for the function arguments are specified by finite intervals. This gives rise to the notion of inclusion function. To be explicit, let

$$f(x_1, \ldots, x_n)$$

denote a function and the domains of its arguments be given by intervals $X_1, \ldots, X_n$. We then define a *lower bound function*

$$f^{\vdash}(x_1, \ldots, x_n) := \min\{f(x_1, \ldots, x_n) \,|\, x_1 \in X_1, \ldots, x_n \in X_n\}$$

an *upper bound function*

$$f^{\dashv}(x_1, \ldots, x_n) := \max\{f(x_1, \ldots, x_n) \,|\, x_1 \in X_1, \ldots, x_n \in X_n\}$$

and a *range function*

$$f^{\vdash\!\dashv}(x_1, \ldots, x_n) := [\,f^{\vdash}(x_1, \ldots, x_n), f^{\dashv}(x_1, \ldots, x_n)\,]$$

The word *inclusion function* is used to refer to any interval-valued function that includes the range function.

Notice that it is most often not possible to find the range function by simply treating the function arguments as intervals and then evaluating the resulting interval expression. This will result in an inclusion function but most often not in the range function. In general, in order to find the range function, one will need a global optimization procedure. This will be discussed in 8.4.

8.3.2 Evaluation of Inclusion Functions

Given an interval function one can use the `evalfi` command to calculate an inclusion function (for terminology see 8.3.1). The syntax is shown in the following box.

```
evalfi(
    arg1=l1,u1,          domain for argument arg1, def. [0,0]
    fmt=...,             print format, def. 12.4
) = interval_function;
```

On the right-hand side must be given the definition of an interval function. Domains for its arguments can be specified as parameters of the command. Of course, `arg` must be substituted by the name of the argument that is used in the function definition. If the function has more than one argument, one can specify a separate domain for each argument.

Interval-valued Gradient. One can use the `evalfi1` command, instead of `evalfi`, in order to calculate an inclusion function also for the gradient of the function specified on the right-hand side.

Example 1 To illustrate we use the function

$$f(x) := \sin(x)$$

Box 1 shows the `evalfi1` command and its result. Since the interval that results for the gradient does not contain zero one can conclude that the function is, in the given interval, monotone.

Box 1 Illustration of `evalfi` command

```
Command:    evalfi1(
                x = 0,1,
            ) = sin(x);

Result:

[      0.0000 ,      0.8415 ]   function value
[      0.5403 ,      1.0000 ]   gradient x
```

8.4 Range of Interval Functions

This section deals with the problem how to find the range, i.e., the global maximum and minimum, of a function that contains interval-valued arguments. Subsections are as follows.

8.4.1 The gmin Command

This section describes the `gmin` command that can be used to find the global minimum of a function. Assume we are given a real-valued function

$$f(x_1, \ldots, x_n)$$

and intervals, $X_1, \ldots, X_n$, that specify domains for the function arguments. The command tries to find

$$f^* := \min\{f(x_1, \ldots, x_n) \,|\, x_1 \in X_1, \ldots, x_n \in X_n\}$$

In order to describe the algorithm we use the following abbreviations:

$$\begin{aligned} x &:= (x_1, \ldots, x_n) \\ X &:= X_1 \times \cdots \times X_n \\ f_l^*(B) &:= \min\{f(x) \,|\, x \in B\} \\ f_u^*(B) &:= \max\{f(x) \,|\, x \in B\} \end{aligned}$$

where $B \subseteq X$. We also use $f_m(B)$ to denote the value of f evaluated at the midpoint of B.

The algorithm keeps a list of boxes, indexed by k. The initial box equals X, the domain of the function. The algorithm tries to find a smallest set of boxes such that the arguments where the function has its global minima are contained in those boxes. If B_k is a box from the list, $I(B_k)$ is used to denote its state. $I(B_k) = 0$ when the box is created. $I(B_k) = 1$ if $W(B_k) \leq T_b$, where $W(B_k)$ is the box width (= maximal width of its marginal intervals) and T_b is a given tolerance for the minimal box width that can be specified with the `tolbw` parameter of the `gmin` command.

(1) Set: $k = 1$, $B_1 = X$. Add B_1 to the (previously empty) list of boxes. Calculate f with the given starting values and denote the result by f^*. Find an inclusion function for $f(B_1)$ in order to get values $f_l(B_1) \leq f_l^*(B_1)$ and $f_u(B_1) \geq f_u^*(B_1)$.

(2) In the current list of boxes, drop all boxes B_k if $f_l(B_k) > f^*$. (These boxes cannot contain global minima.)

(3) In the current list of boxes, find a box B_k such that $I(B_k) = 0$ and $f_l(B_k)$ is minimal. If such a box cannot be found continue with step 11.

(4) If $W(B_k) \leq T_b$ set $I(B_k) = 1$ and continue with step 3.

(5) Bisect B_k into two sub-boxes, B_{k_1} and B_{k_2}.

(6) If it is not requested to use first derivatives (`ns=0`), calculate, for both boxes, an inclusion function for f and put the boxes, together with the lower and upper bounds of the inclusion functions, on the list. Continue with step 8.

(7) Otherwise, if (`ns=1`), calculate, for both boxes, an inclusion function for f and for its gradient. Then, for both boxes, if the inclusion function for the gradient does not contain zero, conclude that the function is monotone in the box and shrink the box to that endpoint where the function takes its minimal value and update the lower and upper bounds for the inclusion function of the box. Then put the boxes, together with the lower and upper bounds of the inclusion functions, on the list.

(8) For $i = 1, 2$, if $f_u(B_{k_i}) - f_l(B_{k_i}) < T_f$, set $I(B_{k_i}) = 1$. (The tolerance T_f can be specified with the `tolfd` parameter of the `gmin` command.)

(9) Set f^* to the minimum of its current value and the minimum of $f_m(B_{k_1})$ and $f_m(B_{k_2})$.

(10) Increase the iteration counter. If the maximum number of iterations is not yet reached, continue with step 2. (The maximum number of iterations can be specified with the `mxit` parameter of the `gmin` command.)

(11) In the current list of boxes, for each box B_k that has $I(B_k) = 1$, if $f_l(B_k) \geq f^* - T_e$, then set $I(B_k) = 2$. (The tolerance T_e can be specified with the `tolfe` parameter in the `gmin` command.) Finally, report information about all boxes where $I(B_k) = 2$ in the standard output. If requested, report also information about boxes where $I(B_k) = 1$ in the protocol file.

The syntax of the `gmin` command is shown in Box 1. The right-hand side must provide a function expression that can be evaluated as an interval expression. Note, however, that the expression must not contain elementary interval expressions.

Box 1 Syntax of `gmin` command

```
gmin(
   ns=...,        ns=1 for using derivatives, def. 0
   xp=...,        directly defined starting values
   dsv=...,       starting values given by data file
   mxit=...,      maximum number of iterations, def. 100
   nbox=...,      number of boxes, def. 100
   tolbw=...,     tolerance for final box width, def. 1.e-6
   tolfd=...,     tolerance for function values, def. 1.e-10
   tolfe=...,     tolerance for convergence, def. 1.e-10
   prot=...,      protocol file
   fmt=...,       print format, def. 13.6
) = function;
```

Starting values, and domains, for the function arguments can be specified with the `xp` parameter in the following way:

$$\text{xp} = x_1[l_1, u_1], x_2[l_1, u_1], \ldots$$

where x_i is the starting value, and $[l_i, u_i]$ is the domain, for the ith argument. Alternatively, one can use the `dsv` parameter to specify a data file that contains starting values in its first column, and lower and upper bounds for the domains in its second and third columns, respectively.

Example 1 To illustrate we try to find the global minima of $\sin(x)$ in the interval $[0, 12]$. Box 2 shows the `gmin` command and part of its standard output.

Box 2 Illustration of gmin command

```
Command:    gmin(
                xp = 1[0,10],
                ns = 1,
            ) = sin(x);

Part of standard output:

Number of iterations performed: 39
Number of function evaluations: 41
Number of inclusion function evaluations: 77
Number of boxes used: 3
Number of temporarily accepted boxes: 2
Number of finally accepted boxes: 2

Best minimal function value: -1.000000   best lower bound: -1.000000

Box  Acc  Width             Lower function bound  Upper function bound
  1    2  2.2888183594e-05 -1.00000000000000e+00 -9.99999999910710e-01
  2    2  1.1444091797e-05 -1.00000000000000e+00 -9.99999999941891e-01

Box  Idx  Parameter
  1    1  x                 4.712379      4.712402

Box  Idx  Parameter
  2    1  x                10.995564     10.995575
```

8.4.2 The range Command

The `range` command is intended to calculate the range of a user-defined function. The syntax is shown in Box 1. The command is basically identical with the `gmin` command described in **8.4.1**. In fact, the `range` command simply calls the `gmin` algorithm two times in order to find both, a global minimum and a global maximum of the function.

Box 1 Syntax of `range` command

```
range (
   ns=...,        ns=1 for using derivatives, def. 0
   xp=...,        directly defined starting values
   dsv=...,       starting values given by data file
   mxit=...,      maximum number of iterations, def. 100
   nbox=...,      number of boxes, def. 100
   tolbw=...,     tolerance for final box width, def. 1.e-6
   tolfd=...,     tolerance for function values, def. 1.e-10
   tolfe=...,     tolerance for convergence, def. 1.e-10
   prot=...,      protocol file
   fmt=...,       print format, def. 13.6
) = function;
```

8.5 Distribution Functions

This section deals with the problem how to find distribution functions for set-valued and, in particular, interval-valued variables. The subsections are as follows.

8.5.1 Set-valued Discrete Variables describes the `sddf` and and `iddf` commands that find distribution functions for set-valued variables.

8.5.2 Interval-valued Variables describes the `idf` commands that calculates distribution functions for interval-valued variables.

8.5.1 Set-valued Discrete Variables

Let $\mathcal{X} = \{1, \ldots, M\}$ denote the property space of a discrete variable that has M possible categories. Let

$$X : \Omega \longrightarrow \mathcal{P}(\mathcal{X})$$

denote a set-valued variable. Assume we are given a set of n observations, o_i, $i = 1, \ldots, n$. Each observation is a subset of $\mathcal{X}$. Now let $\tilde{x} \subset \mathcal{X}$ be some property set. We then define in a first step:

$$\text{pmin}(o_i, \tilde{x}) := \begin{cases} 1 & \text{if } o_i \subseteq \tilde{x} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{pmax}(o_i, \tilde{x}) := \begin{cases} 0 & \text{if } o_i \cap \tilde{x} = \emptyset \\ 1 & \text{otherwise} \end{cases}$$

These expressions are used in a second step to define lower and upper bounds for the distribution function:

$$\text{Pr}^{\vdash}(\tilde{x}) := \frac{1}{n} \sum_{i=1}^{n} \text{pmin}(o_i, \tilde{x})$$

$$\text{Pr}^{\dashv}(\tilde{x}) := \frac{1}{n} \sum_{i=1}^{n} \text{pmax}(o_i, \tilde{x})$$

We furthermore define two versions of an intermediate distribution function. One version is

$$\bar{\text{Pr}}(\tilde{x}) := \frac{1}{n} \sum_{i=1}^{n} \frac{| o_i \cap \tilde{x} |}{| o_i |}$$

and will be called *mean distribution function*. Another version is defined by the following functional equation:

$$\tilde{\text{Pr}}(\tilde{x}) := \frac{1}{n} \sum_{i=1}^{n} \frac{\tilde{\text{Pr}}(o_i \cap \tilde{x})}{\tilde{\text{Pr}}(o_i)}$$

In general, this equation has no closed solution, but one can try to find a solution with an iterative procedure. If a solution can be found, it will be called a *self-consistent distribution*.

Box 1 Syntax of `sddf` command

```
sddf (
   opt=...,                 option, def. 1
                            1 = min, max, and mean distr. function
                            2 = additionally self-cons. distr. function
   mxit=...,                maximum number of iterations, def. 50
   tolf=...,                tolerance for convergence, def. 1.e-3
   prot=...,                protocol file
   df=...,                  optional output file
) = set_valued_variable;
```

The `sddf` Command. The `sddf` command calculates $\Pr^{\vdash}$, $\Pr^{\dashv}$, $\bar{\Pr}$, and, optionally, $\tilde{\Pr}$. The syntax is shown in Box 1. All parameters, except for the definition of a set-valued variable on the right-hand side, are optional. The variable must be specified by a set of indicator variables. In general, if there are M possible categories, one has to specify M indicator variables, say,

$$X_1, \ldots, X_M$$

Then, for each case i in the data matrix, if $X_j = 1$, it will be assumed that o_i contains the jth category, and if $X_j = 0$, it will be assumed that o_i does not contain the jth category.

By default (`opt = 1`), the command only calculates the minimal, maximal, and mean distribution function. If `opt = 2`, the command furthermore tries to find a self-consistent distribution function with an iterative procedure. In this case one can specify a maximum number of iterations with the `mxit` parameter, and a convergence tolerance with the `tolf` parameter. If requested with the `prot` parameter, information about the iterations are written in a protocol file. By default, a table that shows the values of the variable and corresponding values of the distribution functions is written into the standard output. Alternatively, the table is written into an output file that can be specified with the `df` parameter.

Example 1 An illustration is given in Box 2. In this example, $M = 3$ and the data file, `id1.dat`, provides information about five cases:

$$o_1 = \{1\}, o_2 = \{2\}, o_3 = \{3\}, o_4 = \{1, 2\}, o_5 = \{2, 3\}$$

Box 2 Illustration of `sddf` and `iddf` commands

```
Data file id1.dat                Data file id1a.dat

X1 X2 X3                         XL  XH
--------                         ------
 1  0  0                         1   1
 0  1  0                         2   2
 0  0  1                         3   3
 1  1  0                         1   2
 0  1  1                         2   3

Command file: id1.cf             Command file: id1a.cf

nvar(                            nvar(
    dfile = id1.dat,                 dilfe = id1a.dat,
    X1 = c1,                         XL = c1,
    X2 = c2,                         XH = c2,
    X3 = c3,                     );
);
sddf(                            iddf(
    opt=2,                           opt = 2,
    df = df,                         df = df,
) = X1,,X3;                      ) = XL,XH;

Output file: df

X        min df        max df       mean df       s-c. df
-----------------------------------------------------------
1      0.200000      0.400000      0.300000      0.276596
2      0.200000      0.600000      0.400000      0.446809
3      0.200000      0.400000      0.300000      0.276596
```

The `iddf` Command. This command is basically identical with the `sddf` command. The only difference is in the specification of the set-valued variable. While the `sddf` command allows to specify arbitrary subsets, the `iddf` command requires a definition of intervals. The syntax is

```
iddf(...) = XL,XH;
```

where now `XL` and `XH` are names of variables that provide, respectively, lower and upper bounds of the intervals. For an example see the data file, `id1a.dat`, and the command file, `id1a.cf`, in Box 2.

8.5.2 Interval-valued Variables

Let X denote an interval-valued variable. Assume there are n values, o_i, given by the intervals

$$o_i = [\, l_i, u_i \,] \qquad i = 1, \ldots, n$$

The range of the variable will be defined by

$$[\, l, u \,] \qquad l = \min\{l_i \,|\, i = 1, \ldots, n\},\ u = \max\{u_i \,|\, i = 1, \ldots, n\}$$

The term *induced partition* is used to denote a partition of $[\, l, u \,]$ into sub-intervals given by

$$l = \tau_1, \tau_2, \ldots, \tau_{m-1}, \tau_m = u$$

such that each τ_j coincides with at least one l_i or u_i. Then, for any subset $S \subseteq [\, l, u \,]$, we define in a first step:

$$\mathrm{pmin}(o_i, S) := \begin{cases} 1 & \text{if } o_i \subseteq S \\ 0 & \text{otherwise} \end{cases}$$

$$\mathrm{pmax}(o_i, S) := \begin{cases} 0 & \text{if } o_i \cap S = \emptyset \\ 1 & \text{otherwise} \end{cases}$$

These expressions are used in a second step to define lower and upper bounds for the distribution function:

$$\mathrm{Pr}^{\vdash}(S) := \frac{1}{n} \sum_{i=1}^{n} \mathrm{pmin}(o_i, S)$$

$$\mathrm{Pr}^{\dashv}(S) := \frac{1}{n} \sum_{i=1}^{n} \mathrm{pmax}(o_i, S)$$

We furthermore define two versions of an intermediate distribution function. One version is

$$\bar{\mathrm{Pr}}(S) := \frac{1}{n} \sum_{i=1}^{n} \frac{|\, o_i \cap S \,|}{|\, o_i \,|}$$

Box 1 Syntax of `idf` command

```
idf (
   fmt=...,        print format, def. 13.6
) = XL,XH;
```

and will be called *mean distribution function*. Another version is defined by the following functional equation:

$$\tilde{\Pr}(S) := \frac{1}{n} \sum_{i=1}^{n} \frac{\tilde{\Pr}(o_i \cap S)}{\tilde{\Pr}(o_i)}$$

In general, this equation has no closed solution, but one can try to find a solution with an iterative procedure. If a solution can be found, it will be called a *self-consistent distribution*.

The `idf` Command. Given an interval-valued variable the `idf` command calculates the induced partition, $\tau_1, \ldots, \tau_m$, and for each interval $[\,\tau_1, \tau_j\,]$, calculates $\Pr^{\vdash}(\,[\tau_1, \tau_j]\,)$, $\Pr^{\dashv}(\,[\tau_1, \tau_j]\,)$, and $\bar{\Pr}(\,[\tau_1, \tau_j]\,)$. The syntax is shown in Box 1.

Example 1 To illustrate we use the following arbitrarily chosen income data:

$$1900, 1950, 2000, 2010, 2030, 2050, 2100, 2110, 21402140, 2150$$

We assume that the precision is given by ± 30 DM. The data file is `id2.dat`. Box 2 shows the command file, `id2.cf`, and the standard output from the `idf` command.

Note. Calculation of a self-consistent distribution function is not yet available with the `idf` command.

Box 2 Illustration of `idf` command

```
Command file: id2.cf

nvar(
    dfile = id2.dat,
    XL = c1 - 30,
    XH = c1 + 30,
);
idf = XL,XH;

Standard output:

Idx      Partition  Lower Bound  Upper Bound      Mean DF
  1    1870.000000     0.000000     0.090909     0.000000
  2    1920.000000     0.000000     0.181818     0.075758
  3    1930.000000     0.090909     0.181818     0.106061
  4    1970.000000     0.090909     0.272727     0.166667
  5    1980.000000     0.181818     0.363636     0.196970
  6    2000.000000     0.181818     0.454545     0.257576
  7    2020.000000     0.181818     0.545455     0.348485
  8    2030.000000     0.272727     0.545455     0.409091
  9    2040.000000     0.363636     0.545455     0.454545
 10    2060.000000     0.454545     0.545455     0.515152
 11    2070.000000     0.454545     0.636364     0.530303
 12    2080.000000     0.545455     0.727273     0.560606
 13    2110.000000     0.545455     0.909091     0.651515
 14    2120.000000     0.545455     1.000000     0.712121
 15    2130.000000     0.636364     1.000000     0.787879
 16    2140.000000     0.727273     1.000000     0.848485
 17    2170.000000     0.909091     1.000000     0.984848
 18    2180.000000     1.000000     1.000000     1.000000
```

8.6 Descriptive Statistics

This section describes commands that can be used to calculate simple descriptive statistics for interval-valued variables. The subsections are as follows.

8.6.1 Mean Values describes the `imean` command that calculates lower and upper bounds for the mean of an interval-valued variable.

8.6.2 Variances describes the `ivar` command that calculates lower and upper bounds for the variance of an interval-valued variable.

8.6.1 Mean Values

Given an interval-valued variable, the `imean` command calculates lower and upper bounds for its mean value. The syntax is shown in the following box.

```
imean(
    fmt=...,          print format, def. 10.4
) = XL,XH;
```

The right-hand side must specify two variables to be interpreted, respectively, as lower and upper bounds for the values of the variable. Since the mean is a simple linear function, calculation of bounds for the mean value is quite easy. The lower bound can be calculated from the values of `XL`, the upper bound can be calculated from the values of `XH`.

Example 1 To illustrate we use the data file `id1a.dat` shown in Box 8.5.1-2. The command is simply

```
imean = XL,XH;
```

The lower and upper bounds for the mean value are 1.8 and 2.2, respectively.

8.6.2 Variances

Given an interval-valued variable with values

$$o_i = [\, l_i, u_i \,] \qquad (i = 1, \ldots, n)$$

the `ivar` command calculates

$$V^{\vdash} = \min \left\{ \frac{1}{n} \left(\Sigma_i \left(x_i - (\Sigma_j x_j / n) \right) \right)^2 \mid x_i \in [\, l_i, u_i \,] \right\}$$

$$V^{\dashv} = \max \left\{ \frac{1}{n} \left(\Sigma_i \left(x_i - (\Sigma_j x_j / n) \right) \right)^2 \mid x_i \in [\, l_i, u_i \,] \right\}$$

The syntax is shown in the following box.

```
ivar (
    mxit=...,       maximum number of iterations, def. 100
    nbox=...,       number of boxes, def. 100
    tolbw=...,      tolerance for final box width, def. 1.e-4
    tolfd=...,      tolerance for function values, def. 1.e-6
    tolfe=...,      tolerance for convergence, def. 1.e-6
    prot=...,       protocol file
    fmt=...,        print format, def. 10.6
) = XL,XH;
```

The right-hand side must specify two variables to be interpreted, respectively, as lower and upper bounds for the values of the variable. Bounds for the variance are calculated by calling the `gmin` command with a pre-defined expression for the variance; see **8.4.1** for an explanation of parameters.

Example 1 To illustrate we use the data file `id1a.dat` shown in Box **8.5.1**-2. The command is simply

```
ivar = XL,XH;
```

The lower and upper bounds for the variance are 0.4 and 0.8, respectively.

9. References

Abbott, A. [1983]. Sequences of Social Events: Concepts and Methods for the Analysis of Order in Social Processes. Historical Methods 16, 129 – 147.

Abbott, A. [1995]. Sequence Analysis: New Methods for Old Ideas. Annual Review of Sociology 21, 93 – 113.

Abbott, A. [1995a]. A Comment on "Measuring the Agreement Between Sequences". Sociological Methods & Research 24, 232 – 243.

Abbott, A., Hrycak, A. [1990]. Measuring Resemblance in Sequence Data: An Optimal Matching Analysis of Musician's Careers. American Journal of Sociology 96, 144 – 185.

Abramowitz, M., Stegun, I. A. (eds.) [1964]. Handbook of Mathematical Functions. New York: Dover Publications.

Adobe Systems Incorporated [1990]. PostScript Language Reference Manual (2nd edition). New York: Addison-Wesley.

Aitkin, M., Anderson, D., Francis, B., Hinde, J. [1989]. Statistical Modelling in GLIM. Oxford: Clarendon.

Akaike, H. [1973]. Information Theory and an Extension of the Maximum Likelihood Principle. In: Proceedings of the 2nd International Symposium of Information Theory, ed. by B. N. Petrov, F. Csaki. Budapest 1973, pp. 267 – 281.

Akima, H. [1972]. Interpolation and Smooth Curve Fitting Based on Local Procedures (ACM Algorithm 433). Communications of the ACM 15, 914 – 918.

Allison, P. D. [1982]. Discrete-Time Methods for the Analysis of Event Histories. In: Sociological Methodology 1982, ed. by S. Leinhardt, pp. 61 – 98. San Francisco: Jossey-Bass.

Anderberg, M. R. [1973]. Cluster Analysis for Applications. New York: Academic Press.

Andreß , H.-J. [1985]. Multivariate Analyse von Verlaufsdaten. ZUMA-Methodentexte, Band 1. Mannheim: ZUMA.

Andreß , H.-J. [1992]. Einführung in die Verlaufsdatenanalyse. Köln: Zentrum für historische Sozialforschung.

Anton, H., Rorres, C. [1991]. Elementary Linear Algebra. Applications Version. New York: Wiley.

Arabie, P., Hubert, L. J., Schleutermann, S. [1990]. Blockmodels from the Bond Energy Approach. Social Networks 12, 99 – 126.

Baker, L. [1992]. C Mathematical Function Handbook. New York: McGraw-Hill.

Baldone, S., Brioschi, F., Paleari, S. [1997]. Ownership Measures Among Firms Connected by Cross-Shareholdings and a Further Analogy with Input-Output Theory. Mimeo.

Barrodale, I., Roberts, F. D. K. [1973]. Algorithm 478: Solution of an Overdetermined System of Equations in the L_1 Norm. Communications of the ACM 17, 319 – 320.

Barthélemy, J.-P., Guénoche, A. [1991]. Trees and Proximity Representations. New York: Wiley.

Beasley, J. D., S. G. Springer [1977]. The Percentage Points of the Normal Distribution (AS 111). Applied Statistics 26, 118 – 121.

Bell, J. R. [1968]. Normal Random Deviates (Algorithm 334). Communications of the ACM 11, 498.

Bell, M., Pike, M. C. [1966]. Remark on Algorithm 178 (Direct Search). Communications of the ACM 9, 684.

Bernardo, J. M. [1976]. Psi (Digamma) Function. Algorithm AS 103. Applied Statistics 25, 315 – 317.

Birkes, D., Dodge, Y. [1993]. Alternative Methods of Regression. New York: Wiley.

Blossfeld, H.-P., Hamerle, A., Mayer, K. U. [1986]. Ereignisanalyse. Statistische Theorie und Anwendungen in den Sozialwissenschaften. Frankfurt/New York: Campus 1986

Blossfeld, H.-P., Hamerle, A., Mayer, K. U. [1989]. Event History Analysis. Statistical Theory and Applications in the Social Sciences. Hillsdale, NJ: Lawrence Erlbaum.

Blossfeld, H.-P., Klijzing, E., Pohl, K., Rohwer, G. [1996]. Modellierung paralleler und interdependenter Prozesse in der Bevölkerungswissenschaft. Zeitschrift für Bevölkerungswissenschaft 21, 29 – 56.

Blossfeld, H.-P., Rohwer, G. [1995]. Techniques of Event History Modeling. New Approaches to Causal Analysis. Mahwah, NJ: Lawrence Erlbaum.

Bock, H.-H. [1984]. Distanzmaße zum Vergleich von Bäumen, Hierarchien und Sequenzen. In: H.-H. Bock (ed.): Studien zur Klassifikation, Vol. 15 (= Anwendungen der Klassifikation: Datenanalyse und numerische Klassifikation), pp. 52 – 67, Frankfurt: Indeks Verlag.

Boor, C. de [1978]. A Practical Guide to Splines. Berlin and New York: Springer-Verlag.

Born, G. [1995]. Referenzhandbuch Dateiformate. Bonn: Addison-Wesley.

Brent, R. P. [1974]. A Gaussian Pseudo-Random Number Generator. Collected Algorithms from ACM, 488.

Breslow, N. [1974]. Covariance Analysis of Censored Survival Data. Biometrics 30, 89 – 99.

Bron, C., Kerbosch, J. [1973]. Algorithm 457 – Finding All Cliques of an Undirected Graph. Communications of the ACM 16, 575 – 577.

Brüderl, J. [1991]. Bell-shaped Duration Dependence in Social Processes. A Generalized Log-logistic Rate Model. Unpubl. manuscript, Bern.

Brüderl, J., Diekmann, A. [1995]. The Log-logistic Rate Model. Two Generalizations with an Application to Demographic Data. Sociological Methods & Research 24, 158 – 186.

Buckley, J., James, I. [1979]. Linear Regression with Censored Data. Biometrika 66, 429 – 436.

Budde, M., Wargenau, M. [1984]. Analyse von Überlebensdaten in SAS und BMDP. Statistical Software Newsletter 10, 126 – 135.

Burt, R. S. [1978]. Cohesion versus Structural Equivalence as a Basis for Network Subgroups. Sociological Methods & Research 7, 189 – 212.

Cameron, A. C., Trivedi, P. K. [1986]. Econometric Models Based on Count Data: Comparisons and Applications of Some Estimators and Tests. Journal of Applied Econometrics 1, 29 – 53.

Carpaneto, G., Toth, P. [1980]. Solution of the Assignment Problem. ACM Transactions on Mathematical Software 6, 104 – 111.

Chambers, J. M., Cleveland, W. S., Kleiner, B., Tukey, P. A. [1983]. Graphical Methods for Data Analysis. Pacific Grove: Wadsworth & Brooks.

Chan, T. F., Lewis, J. G. [1979]. Computing Standard Deviations: Accuracy. Communications of the ACM 22, 526 – 531.

Chhikara, R. S., Folks, J.L. [1977]. The Inverse Gaussian Distribution as a Lifetime Model. Technometrics 19, 461 – 468.

Chow, T., Eskow, E., Schnabel, R. [1994]. Algorithm 739: A Software Package for Unconstrained Optimization Using Tensor Methods, ACM Transactions on Mathematical Software 20, 518 – 530.

Coale, A. J., McNeil, D. R. [1972]. The Distribution by Age of the Frequency of First Marriage in a Female Cohort. Journal of the American Statistical Association 67, 743 – 749.

Collett, D. [1991]. Modelling Binary Data. London: Chapman & Hall.

Corter, J. E. [1996]. Tree Models of Similarity and Association. Thousand Oakes: Sage.

Cox, D. R. [1972]. Regression Models and Life-Tables. Journal of the Royal Statistical Society 34, 187 – 220.

Cox, D. R. [1975]. Partial Likelihood. Biometrika 62, 269 – 276.

Daganzo, C. [1979]. Multinomial Probit. The Theory and its Application to Demand Forecasting. New York: Academic Press.

Deutsch, J., Flückiger, Y., Silber, J. [1994]. Measuring Occupational Segregation. Journal of Econometrics 61, 133 – 146.

Diekmann, A. [1990]. Diffusion and Survival Models for the Process of Entry into Marriage. In: K. U. Mayer, N. B. Tuma (eds.): Event History Analysis in Life Course Research, pp. 170 – 183. Madison (Wisconsin): University of Wisconsin Press.

Diekmann, A., Mitter, P. [1983]. The "Sickle-Hypothesis": A Time-Dependent Poisson Model with Applications to Deviant Behavior and Occupational Mobility. Journal of Mathematical Sociology 9, 85 – 101.

Diekmann, A., Mitter, P. [1984]. A Comparison of the "Sickle Function" with Alternative Stochastic Models of Divorce Rates, In: Stochastic Modelling of Social Processes, ed. by A. Diekmann, P. Mitter, pp. 123 – 153. New York: Academic Press.

Dielman, T. E. [1984]. Least Absolute Value Estimation in Regression Models: An Annotated Bibliography. Communications in Statistics: Theory and Methods 13, 513 – 541.

Dielman, T., Pfaffenberger, R. [1982]. LAV (Least Absolute Value) Estimation in Linear Regression: A Review. TIMS / Studies in the Management Sciences 19, 31 – 52.

Dierckx, P. [1975]. An Algorithm for Smoothing, Differentiation and Integration of Experimental Data Using Spline Functions. Journal of Computational and Applied Mathematics 1, 165 – 184.

Dijkstra, W., Taris, T. [1995]. Measuring the Agreement Between Sequences. Sociological Methods & Research 24, 214 – 231.

Dixon, W. J. (ed.) [1990]. BMDP Statistical Software Manual, Volume 2. Berkeley: University of California Press.

Donnelly, T. G. [1971]. Bivariate Normal Distribution. Collected Algorithms of ACM, No. 462.

Dorrer, E. [1968]. F-Distribution (Algorithm 322). Communications of the ACM 11.

Drezner, A. [1992]. Computation of the Multivariate Normal Integral. ACM Transactions on Mathematical Software 18, 470 – 480.

Eberlein, P. J., Boothroyd,J. [1971]. Solution to the Eigenproblem by a Norm Reducing Jacobi Type Method. In: J. H. Wilkinson, C. Reinsch, eds., Linear Algebra, pp. 327 – 338. New York: Springer-Verlag.

Eddy, W. F. [1977]. A New Convex Hull Algorithm for Planar Sets. ACM Transactions on Mathematical Software 3, 398 – 403.

Edmonds, J., Karp, R. M. [1972]. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. Journal of the ACM 19, 248 – 264.

Everitt, B. S., Dunn, G. [1991]. Applied Multivariate Data Analysis. London: Arnold.

Falk, M., Becker, R., Marohn, F. [1995]. Angewandte Statistik mit SAS. Berlin: Springer-Verlag.

Feo, T. A., Resende, M. G. C., Smith, S. H. [1994]. A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. Operations Research 42, 860 – 878.

Fletcher, R. [1987]. Practical Methods of Optimization (2nd ed). New York: Wiley.

Floyd, R. W. [1962]. Algorithm 97 – Shortest Path. Communications of the ACM 5, 345.

Francis, B., Green, M., Payne, C. (eds.) [1994]. The GLIM System. Release 4 Manual. Oxford: Clarendon.

Freeman, L. C. [1978]. Centrality in Social Networks. Conceptual Clarification. Social Networks 1, 215 – 239.

Freeman, L. C., Borgatti, S. P., White, D. R. [1991]. Centrality in Valued Graphs: A Measure of Betweennes Based on Network Flow. Social Networks 13, 141 – 154.

Gail, M. [1975]. A Review and Critique of Some Models Used in Competing Risk Analysis. Biometrics 31, 209 – 222.

Gallant, A. R. [1987]. Nonlinear Statistical Models. New York: Wiley.

Galton, A. [1984]. The Logic of Aspect. Oxford: Clarendon.

Garribba, S., Quartapelle, L., Reina, G. [1978]. SNIFF: Efficient Self-Tuning Algorithm for Numerical Integration. Computing 20, 363 – 375.

Gehan, E. A. [1965]. A Generalized Wilcoxon Test for Comparing Arbitrarily Singly-Censored Samples. Biometrika 52, 203 – 223.

Gentleman, W. M., Marovich, S. B. [1974]. More on Algorithms that Reveal Properties of Floating point Arithmetic Units. Communications of the ACM 17, 276 – 277.

Gibbons, A. [1985]. Algorithmic Graph Theory. Cambridge: Cambridge University Press.

Gibbs, N. E. [1969]. A Cycle Generation Algorithm for Finite Undirected Linear Graphs. Journal of the ACM 16, 564 – 568.

Golub, G. H, Reinsch, C. [1971]. Singular Value Decomposition and Least Squares Solutions. In: J. H. Wilkinson, C. Reinsch, eds., Linear Algebra, pp. 134 – 151. New York: Springer-Verlag.

Goodall, C. [1990]. A Survey of Smoothing Techniques. In: J. Fox, J. S. Long (eds.), Modern Methods of Data Analysis, pp. 126 – 176. Newbury Park: Sage.

Grad, J, Brebner, M. A. [1963]. Eigenvalues and Eigenvectors of a Real General Matrix. Algorithm 343 from CACM. Communications of the ACM 11, 820–26.

Greene, W. H. [1991]. Econometric Analysis. New York: Macmillan.

Greene, W. H. [1992]. LIMDEP. User's Manual and Reference Guide, Version 6.0. Bellport, NY: Econometric Software Inc.

Greene, W. H., Seaks, T. G. [1991]. The Restricted Least Squares Estimator: A Pedagogical Note. Review of Economics and Statistics 73, 563 – 567.

Guénoche, A., Hansen, P., Jaumard, B. [1991]. Efficient Algorithms for Divisive Hierarchical Clustering with the Diameter Criterion. Journal of Classification 8, 5 – 30.

Haberman, S. J. [1978]. Analysis of Qualitative Data. 2 Vols. New York: Academic Press.

Härdle, W. [1991]. Smoothing Techniques. With Implementation in S. New York: Springer-Verlag.

Haisken-DeNew, J. P., Schmidt, C. M. [1997]. Inter-Industry and Inter-Region Differentials: Mechanics and Interpretation. Review of Economics and Statistics, 79.

Hamerle, A. [1989]. Multiple-Spell Regression Models for Duration Data. Applied Statistics 38, 127 – 138.

Hamerle, A., Tutz, G. [1989]. Diskrete Modelle zur Analyse von Verweildauer und Lebenszeiten. Frankfurt: Campus.

Hansen, E. 1992. Global Optimization Using Interval Analysis. New York: Marcel Dekker.

Hanson, R. J., Haskell, K. H. [1982]. Two Algorithms for the Linearly Constrained Least Squares Problem (Algorithm 587). ACM Transactions on Mathematical Software 8, 323 – 333.

Harary, F., Ross, I. C. [1957]. A Procedure for Clique Detection Using the Group Matrix. Sociometry 20, 205–215.

Harley, P. J. [1986]. Nonlinear Optimization: Unconstrained. In: Numerical Algorithms, ed. by J.L. Mohamed, J.E. Walsh, pp. 239 – 259. Oxford: Clarendon.

Heijden, P. G. M. van der [1987]. Correspondence Analysis of Longitudinal Categorical Data. Leiden: DSWO Press.

Heijden, P. G. M. van der, Leeuw, J. de [1989]. Correspondence Analysis, with Special Attention to the Analysis of Panel Data and Event History Data. In: C. C. Clogg (ed.): Sociological Methodology 1989 (Vol. 19), pp. 43 – 87. San Francisco: Jossey-Bass.

Hernes, G. [1972]. The Process of Entry into First Marriage. American Sociological Review 37, 173 – 182.

Hill, I, D. [1973]. The Normal Integral (AS 66). Applied Statistics 22, 424 – 427.

Hill, I. D., Pike, M. C. [1967]. Chi-Squared Integral (Algorithm 299). Communications of the ACM 10, 243 – 244.

Hill, G. W., Davis, A. W. [1971]. Normal Deviate. Collected Algorithms from CACM, No. 442.

Hubert, L. [1973]. Monotone Invariant Clustering Procedures. Psychometrika 38, 47 – 62.

Huinink, J., Henz, U. [1993]. Probleme der parametrischen Analyse des Alters bei der Familiengründung und der Schätzung von Geburtenabständen. In: A. Diekmann, S. Weick (eds.): Der Familienzyklus als sozialer Prozeß, pp. 259 – 284. Berlin: Duncker & Humblot.

Jain, A. K., Dubes, R. C. [1988]. Algorithms for Clustering Data. Englewood Cliffs: Prentice Hall.

James, D. R., Taeuber, K. E. [1985]. Measures of Segregation. In: N. B. Tuma (ed.): Sociological Methodology 1985, pp. 1 – 32. San Francisco: Jossey-Bass.

Johnson, S. C. [1967]. Hierarchical Clustering Schemes. Psychometrika 32, 241 – 254.

Johnston, J. [1972]. Econometric Methods (2nd ed.). London: McGraw-Hill.

Judge, G. G., Griffiths, W. E., Hill, R. C., Lee, T.-C. [1980]. The Theory and Practice of Econometrics. New York: Wiley.

Judge, G. G., Hill, R. C., Griffiths, W. E., Lütkepohl, H., Lee, T.-C. [1988]. Introduction to the Theory and Practice of Econometrics, 2nd ed. New York: Wiley.

Kalbfleisch, J. D., Prentice, R.L. [1980]. The Statistical Analysis of Failure Time Data. New York: Wiley.

Kaplan, E. L., Meier, P. [1958]. Nonparametric Estimation from Incomplete Observations. Journal of the American Statistical Association 53, 457 – 481.

Kaupe, A. F. [1963]. Direct Search (Algorithm 178). Communications of the ACM 6, 313.

Kendall, M., Stuart, A. [1977]. The Advanced Theory of Statistics, Vol. 1. London: Charles Griffin & Comp.

Kruskal, J. B. [1983]. An Overview of Sequence Comparison. In: D. Sankoff, J.B. Kruskal (eds.), Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, pp. 1–44, Reading: Addison-Wesley.

Kruskal, J. B., Sankoff, D. [1983]. An Anthology of Algorithms and Concepts for Sequence Comparisons. In: D. Sankoff, J.B. Kruskal (eds.), Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, pp. 265 – 310. Reading: Addison-Wesley.

Lancaster, T. [1985]. Generalised Residuals and Heterogeneous Duration Models. Journal of Econometrics 28, 155 – 169.

Lancaster, T. [1990]. The Econometric Analysis of Transition Data. Cambridge: University Press.

Lau, C.-L. [1980]. A Simple Series for the Incomplete Gamma Integral (AS 147). Applied Statistics 29, 113 – 114.

Lawless, J. F. [1980]. Inference in the Generalized Gamma and Log Gamma Distributions. Technometrics 22, 409 – 419.

Lawless, J. F. [1982]. Statistical Models and Methods for Lifetime Data. New York: Wiley.

Lawson, C. L., Hanson, R. J. [1974]. Solving Least Squares Problems. Englewood Cliffs: Prentice-Hall.

Leathers, B. L. [1977]. Tabulating Sparse Joint Frequency Distributions, Algorithm AS 119. Applied Statistics 26, 364 – 368.

Lee, E. T. [1980]. Statistical Methods for Survival Data Analysis. Belmont, CA.

Lenstra, J. K. [1974]. Clustering a Data Array and the Travelling Salesman Problem. Operations Research 22, 413 – 414.

Lenstra, J. K., Kan, A. H., Rinnooy, G. [1975]. Some Simple Applications of the Travelling Salesman Problem. Operations Research 22, 717 – 733.

Levine, D. A. [1969]. Student's t-Distribution (Algorithm 344). Communications of the ACM 12, 37 – 38.

Liang, C.-K. [1993]. An $O(n^2)$ Algorithm for Finding the Compact Sets of a Graph. BIT 33, 390 – 395.

Long, J. S. [1987]. A Graphical Method for the Interpretation of Multinomial Logit Analysis. Sociological Methods & Research 15, 420 – 446.

Lorrain, F., White, H. C. [1971]. Structural Equivalence of Individuals in Social Networks. Journal of Mathematical Sociology 1, 49 – 80.

Ludwig-Mayerhofer, W. [1990]. Multivariate Logit-Modelle für ordinalskalierte abhängige Variablen. ZA-Informationen 27, pp. 62 – 88. Köln: Zentralarchiv für empirische Sozialforschung.

Lustbader, E. D., Stodola, R. K. [1981]. Partial and Marginal Association in Multidimensional Contingeny Tables (Algorithm AS160). Applied Statistics 30, 97–105.

Malcolm, M. A. [1972]. Algorithms to Reveal Properties of Floating-point Arithmetic. Communications of the ACM 15, 949 – 951.

Mardia, K. V., Kent, J. T., Bibby, J. M. [1979]. Multivariate Analysis. New York: Academic Press.

Martin, R. S., Peters, G., Wilkinson, J. H. [1971]. Symmetric Decomposition of a Positive Definite Matrix. In: J. H. Wilkinson, C. Reinsch, eds., Linear Algebra, pp. 9 – 30. New York: Springer-Verlag.

Mayer, K. U., Brückner, E. [1989]. Lebensverläufe und Wohlfahrtsentwicklung. Konzeption, Design und Methodik der Erhebung von Lebensverläufen der Geburtsjahrgänge 1929 – 1931, 1939 – 1941, 1949 – 1951. Parts I, II, III. Berlin: Max-Planck-Institut für Bildungsforschung.

Mayer, K. U., Huinink, J. [1990]. Age, Period, and Cohort in the Study of the Life Course: A Comparison of Classical APC-Analysis with Event History Analysis. In: D. Magnusson, L. R. Bergman (eds.), Data Quality in Longitudinal Research, pp. 211 – 232. Cambridge: Cambridge University Press.

McCormick, G. P. [1983]. Nonlinear Programming. Theory, Algorithms, and Applications. New York: Wiley.

McCormick, W. T., Schweitzer, P. J., White, T. W. [1972]. Problem Decomposition and Data Reorganization by a Clustering Technique. Operations Research 20, 993 – 1009.

McCullagh, P., Nelder, J. A. [1983]. Generalized Linear Models. London: Chapman and Hall.

McIlroy, M. D. [1969]. Generator of Spanning Trees (Algorithm 354). Communications of the ACM 12, 511.

McKelvey, R. D., Zavoina, W. [1975]. A Statistical Model for the Analysis of Ordinal Level Dependent Variables. Journal of Mathematical Sociology 4, 103 – 120.

Mohamed, J. L., Wait, R. [1986]. Solution of Linear Equations. In: Numerical Algorithms, ed. by J. L. Mohamed, J. E. Walsh, pp. 1 – 28. Oxford: Clarendon.

Moore, R. J. [1982]. Derivatives of the Incomplete Gamma Integral (AS 187). Applied Statistics 31, 330 – 335.

Moreau, T., O'Quigley, J., Mesbah, M. [1985]. A Global Goodness-of-fit Statistic for the Proportional Hazards Model. Applied Statistics 34, 212 – 218.

Mortenson, M. E. [1989]. Computer Graphics. An Introduction to the Mathematics and Geometry. Oxford: Heinemann Newnes.

Namboodiri, K., Suchindran, C. M. [1987]. Life Table Techniques and their Applications. New York: Academic Press.

Neely, P. M. [1966]. Comparison of Several Algorithms for Computation of Means, Standard Deviations and Correlation Coefficients. Communications of the ACM 9, 496 – 499.

Nelder, J. A., Mead, R. [1965]. A Simplex Method for Function Minimization. Computer Journal 7, 308 – 313.

Neumann, K., Morlock, M. [1993]. Operations Research. Münnchen: Hanser.

Nikolai, P. J. [1979]. Eigenvectors and Eigenvalues of Real Generalized Symmetric Matrices by Simultaneous Iteration. ACM Transactions on Mathematical Software 5, 118 – 125.

O'Neill, R. [1974]. Function Minimization Using a Simplex Procedure (AS 47). Applied Statistics 23, 338 – 345.

O'Quigley, J., Roberts, A. [1980]. Weibull: A Regression Model for Survival Time Studies. Computer Programs in Biomedicine 12, 14 – 18.

Pape, U. [1980]. Shortest Path Lengths. ACM Transactions on Mathematical Software 6, 450 – 455.

Paton, K. [1969]. An Algorithm for Finding a Fundamental Set of Cycles of a Graph. Communications of the ACM 12, 514 – 518.

Patterson, T. N. L. [1973]. Algorithm for Automatic Numerical Integration Over a Finite Interval (= CACM 468). Communications of the ACM 16, 694 – 699.

Penna, M. A., Patterson, R. R. [1986]. Projective Geometry and its Applications to Computer Graphics. Englewood Cliffs: Prentice-Hall.

Peters, G., Wilkinson, J. H. [1971]. Eigenvectors of Real and Complex Matrices by LR and QR Triangularizations. In: J. H. Wilkinson, C. Reinsch, eds., Linear Algebra, pp. 372 – 395. New York: Springer-Verlag.

Piessens, R., Doncker-Kapenga, E., Überhuber, C. W., Kahaner, D. K. [1983): QUADPACK. A Subroutine Package for Automatic Integration. Berlin: Springer-Verlag.

Pike, M. C., Hill, I. D. [1965]. Pseudo-Random Numbers (Algorithm 266). Communications of the ACM 8, 605 – 606.

Prentice, R. L. [1978]. Linear Rank Tests with Right-Censored Data. Biometrika 65, 167 – 179.

Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T. [1988]. Numerical Recipes in C. Cambridge: Cambridge University Press.

Pringle, R. M., Rayner, A. A. [1971]. Generalized Inverse Matrices with Applications to Statistics. London: Griffin.

Pruitt, R. C. [1993]. Small Sample Comparison of Six Bivariate Survival Curve Estimators. Journal Statist. Comput. Simul. 45, 147 – 167.

Rao, C. R. [1973]. Linear Statistical Inference and Its Applications, New York: Wiley.

Resende, M. G. C., Pardalos, P. M., Li, Y. [1996]. Algorithm 754: Fortran Subroutines for Approximate Solution of Dense Quadratic Assignment Problems Using GRASP. ACM Transactions on Mathematical Software 22, 104 – 118.

Röding, M., Küsters, U., Arminger, G. [1985]. KALOS Programmbeschreibung. Ein interaktives Programmsystem zur Analyse kategorialer Logitmodelle. Köln: Zentralarchiv für Empirische Sozialforschung.

Rohwer, G. [1996]. Zur Konzeption ereignisanalytischer Modelle in der empirischen Sozialforschung. Paper presented at the ZUMA workshop on Panel Analysis, November 1996.

Salazar, R. C., Sen, S. K. [1968]. Minit Algorithm for Linear Programming (Algorithm 333). Communications of the ACM 11, 437 – 440.

Sankoff, D., Kruskal, J. B. (eds.) [1983]. Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison. Reading: Addison-Wesley.

SAS Institute [1985]. SAS User's Guide: Statistics, Version 5 Edition. Cary, NC: SAS Institute Inc.

Schaffer, H. E. [1970]. CACM 369. Communications of the ACM 13.

Schneider, B. E. [1978]. Trigamma Function. Algorithm AS 121. Applied Statistics 27, 97 – 99.

Schneider, H. [1991]. Verweildaueranalyse mit GAUSS. Frankfurt: Campus.

Schnell, R. [1994]. Graphisch gestützte Datenanalyse. München: Oldenbourg.

Schrage, L. [1979]. A More Portable Fortran Random Number Generator. ACM Transactions on Mathematical Software 2, No. 2.

Sedgewick, R. [1990]. Algorithms in C. New York: Addison-Wesley.

Shanno, D. F., Phua, K. H. [1980]. Remark on Minimization of Unconstrained Multivariate Functions (Algorithm 500). ACM Transactions on Mathematical Software 6, 618 – 622.

Smith, L. B. [1969]. Remark on Algorithm 178 (Direct Search). Communications of the ACM 12, 638.

Smith, T. L., Putman, J. E., Gehan, E. A. [1970]. A Computer Program for Estimating Survival Functions from the Life Table. Computer Programs in Biomedicine 1, 58 – 64.

Snell, E. J. [1964]. A Scaling Procedure for Ordered Categorical Data. Biometrics 20, 592 – 607.

Snyder, J. M. [1992]. Interval Analysis for Computer Graphics. Computer Graphics 26, 121 – 130.

Snyder, W. V. [1978]. Contour Plotting (Algorithm 531). ACM Transactions on Mathematical Software 4, 290 – 294.

Späth, H. [1975]. Cluster-Analyse-Algorithmen zur Objektklassifizierung und Datenreduktion. München: Oldenbourg.

Späth, H. [1983]. Cluster-Formation und -Analyse. München: Oldenbourg.

Späth, H. [1992]. Mathematical Algorithms for Linear Regression. New York: Academic Press.

Sparks, D. N., Todd, A. D. [1973]. Latent Roots of a Symmetric Matrix (AS 60). Applied Statistics 22, 260 – 265.

SPSS [1991]. SPSS Statistical Algorithms (2nd ed). Chicago: SPSS Inc.

Suits, D. B. [1984]. Dummy Variables: Mechanics V. Interpretation. Review of Economics and Statistics 66, 177 – 180.

Swan, [1969]. The Reciprocal of Mill's Ratio (AS 17). Applied Statistics 18, 115 – 116.

Tarone, R. E., Ware, J. [1977]. On Distribution-Free Tests for Equality of Survival Distributions. Biometrika 64, 156 – 160.

Theil, H. [1972]. Statistical Decomposition Analysis. Amsterdam: North-Holland.

Thisted, R. A. [1988]. Elements of Statistical Computing. New York: Chapman and Hall.

Tiernan, J. C. [1970]. An Efficient Search Algorithm to Find the Elementary Circuits of a Graph. Communications of the ACM 13, 722 – 726.

Törn, A., Viitanen, S. [1992]. Topographical Global Optimization. In: C. A. Floudas, P. M. Pardalos (eds.), Recent Advances in Global Optimization, pp. 384 – 398. Princeton: Princeton University Press.

Tomlin, F. K., Smith, L. B. [1969]. Remark on Algorithm 178 (Direct Search). Communications of the ACM 12, 637.

Tong, Y. L. [1980]. Probability Inequalities in Multivariate Distributions. New York: Academic Press.

Tong, Y. L. [1990]. The Multivariate Normal Distribution. New York: Springer-Verlag.

Torgerson, W. S. [1965]. Theory and Methods of Scaling. New York: Wiley.

Tuma, N. B. [1980]. Invoking RATE. Mannheim: ZUMA.

Tuma, N. B., Hannan, M. T. [1984]. Social Dynamics. Models and Methods. New York: Academic Press.

Turau, V. [1996]. Algorithmische Graphentheorie. Bonn: Addison Wesley.

Ullmann-Margalit, E., Morgenbesser, S. [1977]. Picking and Choosing. Social Research 44, 757.

Warshall, S. A. [1962]. A Theorem in Boolean Matrices. Journal of the ACM 9, 11 – 12.

Waterman, M. S. [1995]. Introduction to Computational Biology. London: Chapman & Hall.

Wasserman, S., Faust, K. [1994]. Social Network Analysis: Methods and Applications. Cambridge: Cambridge University Press.

Watt, A. [1989]. Fundamentals of Three-Dimensional Computer Graphics. New York: Addison-Wesley.

Wengert, R. E. [1964]. A Simple Automatic Derivative Evaluation Program. Communications of the ACM 7, 463 – 464.

Wessel, P., Smith, W. H. F. [1996]. A global self-consistent hierarchical high-resolution shoreline database. Journal of Geophysical Research 101, 8741–8743.

Wexler, A. S. [1987]. Automatic Evaluation of Derivatives. Applied Mathematics and Computation 24, 19 – 46.

White, H. [1980]. A Heteroscedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroscedasticity. Econometrica 48, 817 – 838.

White, H. [1982]. Maximum Likelihood Estmation of Misspecified Models. Econometrica 50 [1982), 1–25

Winkelmann, R., Zimmermann, K. F. [1991]. A New Approach for Modeling Economic Count Data. Münchener Wirtschaftswissenschaftliche Beiträge. Discussion Paper Nr. 91-17. Universität München.

Wu, L. L. [1990]. Simple Graphical Goodness-of-Fit Tests for Hazard Rate Models. In: K. U. Mayer, N. B. Tuma (eds.), Event History Analysis in Life Course Research, pp. 184 – 199. Madison (Wisconsin): University of Wisconsin Press.

Yamaguchi, K. [1991]. Event History Analysis. Newbury Park: Sage.

10. Index