CMAT©

Tutorial

Extension of C Language:
Matrix Algebra, Statistics,
Nonlinear Optimization and Estimation
Release 9 (September 2016)

Wolfgang M. Hartmann [1]
Friedrich-Ebert-Anlage 46 / I
D-69117 Heidelberg, Germany

February 12, 2018

[1]Thanks to my dear wife Walee and my old brave Apollo 4500.

Contents

List of Tables

Chapter 1

User Software License Agreement

Please read this license carefully before using the software. By installing or using this software, you are agreeing to be bound by the terms of this license. If you do not agree to the terms of this license, either contact the author or promptly remove the software.

1. This software may not be distributed to third parties.
2. Supply of any part of this software as part of another software requires the separate prior written agreement of the CMAT managers, which may include financial terms.
3. This software is copyrighted and may not be modified, decompiled, reverse engineered, or disassembled.
4. Due acknowledgment shall be made of the use of CMAT in research reports or publications.
5. The academically priced version of this software may only be used for teaching or research.

As long as the author has a regular income, this software is distributed as non-profit and patience is expected since *"everything free comes without guarantee"*. The author is grateful for responses by users such as bug reports or proposals for improvement.

Chapter 2

Introduction

CMAT is one more interactive matrix language, similar to (but not a clone of) MATLAB[577], O-Matrix, GAUSS, SAS/IML®[**?**], SCILAB, EULER, OCTAVE, or YORICK. The author created CMAT to have a tool that fits his own needs and those of people like him who need a language which is numerically stable and very efficient in computer time and memory usage.

The following principles dominated the design of CMAT:

- CMAT is an interpreter and compiler. CMAT executes the programming statements in the order in which they are typed. At the same time, the statements are compiled and assembler code is written to an output file. Exceptions of this process are special compound statements, e.g. `switch` or forward `goto`'s where the execution (but not the compilation) of statements can be delayed or suspended. When the closing bracket of the compound statement or the label of the forward jump is found, the whole block of compiled code is executed. Also, when the compiled code segment is passed later again, for example after a backward `goto`, the compiled assembler code is executed. User specified functions must be defined before they can be called. Therefore, functions are always executed by running through prior compiled assembler code. When CMAT is run with standard input (interactively), the output is written to standard output by default but can be redirected to two output files, one for error messages, warnings, and notes, the other for results. If CMAT is run with an input file (in batch mode), the output is written automatically to the two files. Using the `#include` command you can redirect the program input to other files.

- The language offers easy tools to write data objects into temporary or permanent files and to access them by their name in the computations. When referring to the name of the object, CMAT remembers whether the object is stored incore or in a file.

- CMAT's syntax is similar to that of the C language. Users who know how to write C code do not need to look up a syntax manual for the scalar operations. In addition to the interpreter capability, there are, however, some important differences to C. And the language is extended to deal with vectors, matrices, tensors, data lists (containing scalars, vectors, matrices, tensors, structs, and sublists), k dimensional trees. See page 22 for more details.

- CMAT provides an easy interface for almost all subroutines available in LINPACK, ARPACK, EISPACK[772][284], and LAPACK. Additionally, it provides a set of numerical subroutines which are currently only difficult to come by, including some for sparse matrix analysis, linear and nonlinear optimization, and especially nonlinear and robust, statistical estimation methods.

Some of the results reported in this manual depend on the C compiler used and may slightly differ from those obtained by the user. CMAT was developed in two versions:

1. using the Domain C compiler (see [223]) running in BSD 4.3 version of UNIX [129].
2. using the Watcom C/C++ and Fortran 77 compilers running in Windows NT version 4.

CMAT offers an interface to *gnuplot* for graphical output. Currently there are two ways to connect to `gnuplot`:

interactively The `gnuplot { ... }` syntax permits the input of code submitted to the *gnuplot* software.

batch mode The `rc = gpbatch(gpfiles)` function executes *gnuplot* command scripts stored in one or more `.gp` files equivalent to a DOS or Linux call:
`gnuplot gpfil_1 ... gpfil_n`. The `gpfiles` input argument must be either a string scalar or vector of strings with the path names to the files. As with *gnuplot* the `"-"` "file name" refers to an interactive input section.

The user must download a current version of the *gnuplot* software from the internet and unzip it into a `gnuplot` directory parallel to the ĉmat directory.

The use of some terminals (output devices in *gnuplot*) needs downloading of additional software from the internet:

svg needs the `SVGView.exe` software, e.g. downloadable for no charge from Adobe.

epslatex needs the *graphicx* software which is downloadable for no charge from the internet.

Specificially two programs are used from BSD: YACC for parsing and LEX for lexical analysis. The PC version uses *MKS Lex & Yacc* [618]. This manuscript was typeset using LaTeXby PCTeX32 [670].

The author of CMAT would like to thank the following scientists and friends for contributing software and advice:

- ARPACK Team (Lehouq, Sorensen, and Yang)
- LAPACK team, especially J. Dongarra and Z. Bai
- Michael Berry, University of Tennessee
- Åke Björck and Mikael Adlers, Linköping Universitet, Sweden
- Paul Boggs, National Institute of Standards
- Michael Browne, Ohio State University
- Frank Bretz, Hannover University, Germany
- Andreas Christmann, University of Dortmund
- William J. Cook, University of Waterloo, Canada
- Ove Edlund, LuleåUniversity of Technology, Sweden
- John J. Forrest, Cbc and Clp developer retired from IBM Watson
- Alan Genz, Washington State University

- Robert Hartwig, NC State University, Raleigh
- Don Hedeker, University of Illinois, Chicago
- Harry Joe, University of British Columbia
- Linda Kaufman, Lucent Technologies
- Jan de Leeuw, University of California, Los Angeles
- Bernd Luevelsmeyer, HEITEC Erlangen
- Kaj Madsen and H. Nielsen, Technical University of Denmark
- Alberto Maydeu-Olivares, Barcelona and Madrid
- Olvi Mangasarian, University of Wisconsin, Madison
- Rod McDonald, University of Illinois, Champaign
- Jorge Moré, Argonne National Laboratory
- Stephen Nash, George Mason University
- Esmond Ng, Oak Ridge National Laboratory
- Mustafa Pinar, Bilkent University, Turkey
- Michael J. D. Powell, University of Cambridge, UK
- Thomas Ragg, quantiom (programmer of frontend)
- Jim Ramsay, Mc Gill University, Montreal, Canada
- Peter Rousseeuw, University of Antwerp, Belgium
- William W. Rozeboom, University of Alberta, Canada
- Michael Saunders, Stanford University
- Robert Schnabel, University of Colorado
- Paul Somerville, University of Central Florida
- Peter Spelucci, Technical University Darmstadt, Germany
- Yoshio Takane, McGill University, Montreal
- Gail Towne, Winston-Salem NC
- Stephen Wright, Argonne National Laboratory

Any bugs in CMAT should be contributed entirely to its author. Although the author of CMAT is employed at a major software company, the design and implementation of CMAT was done solely in his spare time, and there is no other connection to his employers product than his experience as a programmer. CMAT contains some modules which look very similar to software the author developed as an employee for this software company. However, the code here is developed completely independend. The first version of CMAT was developed on an Apollo 4500 computer purchased at an employee sale in 1993 for the amount of $ 10. Money was needed also for

1. Mortice-Kern Yacc and Lex software.
2. The Latex compiler by PC TeX.
3. The Watcom C/C++ and FORTRAN 77 compilers with documents, which are free of charge in the meantime.
4. Some newer PC hardware and computer supplies, like CD ROM and backup drives and cartridges. Power supply.

Chapter 3

Installing and Running CMAT

3.1 Installing CMAT in MS Windows

3.1.1 Use of the Installer

The installer writes all necessary files into a `cmat` directory and installs Tcl TK for the frontend.

1. Run the file install.exe which is on the CD ROM.
2. Your screen will turn blue and all necessary files are copied to a directory tree `c:\cmat` or another location of your choice.
3. After the file copying process you must define a path for opening the frontend program:
   (a) The frontend program is `c:\cmat\mytest\cmat-fe.tcl`
   (b) Use the *Open with ...* box to define the program `c:\cmat\tcltk\bin\wish83.exe` for opening the frontend file `cmat-fe.tcl`
4. Now the frontend display should be on your screen for the first time. For future use you should create a shortcut icon to `c:\cmat\mytest\cmat-fe.tcl`

See the `README.txt` for a few more details.

3.1.2 Manual Installation in MS Windows

The CD contains the following directory structure:

1. cmat\com: with subdirectory: cmat\com\main: contains the Windows executable `cmat.exe` together with a number of MS Windows DLL's which are needed for running CMAT:

   `xarpack.dll` Arpack ([505])

   `xfsrc.dll` lots of Fortran (public domain) code

   `xlapack.dll` real Lapack subroutines (Fortran version, see [15])

`zlapack.dll` complex Lapack subroutines (Fortran version)

`xfsrc.dll` subroutine library

`xfsrc2.dll` subroutine library

`xfsrc3.dll` subroutine library

`xqhull.dll` subroutine library

`cmatm0.m` contains storable summary information about the message file `cmatm.m`. The content of this file is read into memory during the initialization of CMAT.

`cmatm.m` contains the compact message file as it is used during the CMAT run. The content of this file is not stored into RAM.

See the following sections about the two kinds of CMAT execution: interactively and in batch mode.

2. cmat\doc: contains the following files of the manual:

   `cm_all.pdf` contains the main user manual

   `cm_sum.pdf` contains a short summary

   `cm_tut.tex` contains the tutorial part

   `cm_ref.tex` contains the reference part

   `cm_det.tex` contains the *Details* chapter and the chapter with *Some Useful CMAT Modules* and a chapter with a collection of more or less famous *Data Sets*.

   Note that `cm_all.pdf` contains all information, whereas the other files contain only parts of `cm_all.pdf`. This directory also contains the corresponding `...dvi` files.

3. cmat\save: contains data objects created by the `obj2fil()` function. The names of such files can be used like variables in statements.

4. cmat\tcltk: contains the binary Tcl and Tk files for running the frontend.

5. cmat\mytest: contains the TclTk script of the user frontend. If you are not pleased with its function you may enhance this code. Otherwise this is an almost empty directory designed to be your working playpen. Of course you may create directories of your own choice. However, working in a directory one level beneath the `cmat` directory is preferrable for finding the executable `cmat.exe` in `cmat\com\main`.

6. cmat\test: contains a number of files for testing cmat. The .inp files contain the input statements for CMAT, the .log files contain the log output, and the .txt files contain the text output.

7. cmat\tgplt: contains a number of files for some `gnuplot` applications

8. cmat\tmicro: contains a number of files for the analysis of microchip data

9. cmat\tnlp: contains a number of files for testing the `lp`, `qp`, and `nlp` functions of cmat.

10. cmat\tsem: contains a number of files for testing the `sem` function of cmat.

You are highly recommended to keep the directory structure as it is, especially if you intend to change the message files. For using the Tcl/TK frontend you should perform the the last steps which were described in the section above (3.1.1).

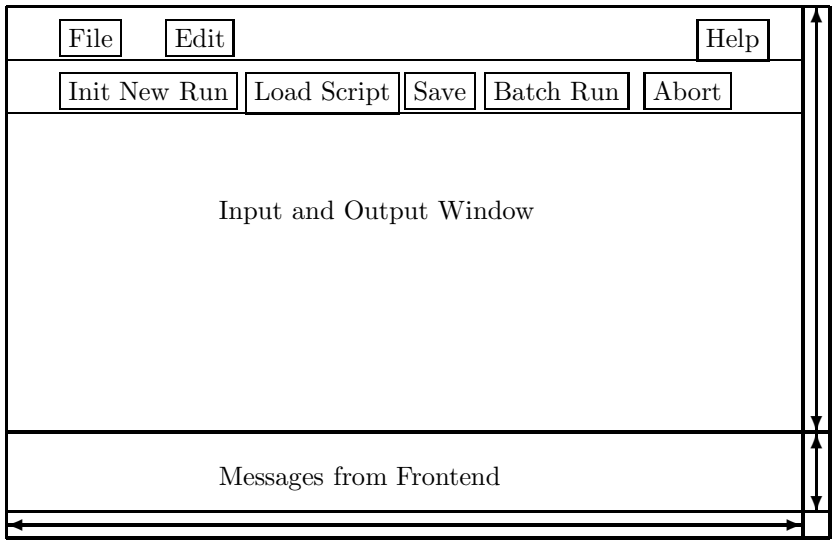

Figure 3.1: Description of Frontend

3.2 Execution of CMAT

The ideal environment to run CMAT is an operating system that provides both, an editor and a shell, e.g. UNIX or Linux. The PC version contains a frontend written in *Tcl TK* which makes interactive and batch processing easier. There are alternatives for the PC, if you don't want to use our frontend:

- For batch processing, even MS DOS would be sufficient, for the interactive processing one of the DOS Extender could be necessary.
- The developers preferred way is to use GNU NTEmacs (working in Windows NT and 95) together with BASH. Working with tools like *MKS Toolkit* would be possible too.

The Linux and Unix operating systems with command line input are more appropriate for CMAT. There will be such versions later in 2003.

3.2.1 Execute CMAT Interactively

The PC version of CMAT comes with a small frontend (GUI) since the command line input with MS DOS is not very comfortable and not many people know how to work with GNU NTEmacs and BASH. The Installer program loads not only the Tcl TK code for the frontend, it will also install the Tcl and TK software itself if it is not already installed at the users computer.

The frontend window is divided into two parts. The upper part is for command line input and all output which normally is written to the `.log` and `.lst` file during batch processing. The lower part contains only output in form of messages from the frontend program. The upper part of the input window contains a row of menus and a row of buttons. The following menus are available:

File tis menu contains only the choices `Load`, `Save`, `UF_Kill`, and `Exit`.

Edit this menu only contains `Clear`.

Help this menu contains choices to open online help documents (summary, tutorial, reference, detail, and complete users manual) and it contains an `About` window button.

The second row contains the following buttons:

Init New Run this will clean the input window and start another CMAT invocation

Load Script this will open a browser window permitting the input of a script which is immediately executed. The input and output is visible in the upper window.

Save for saving the content of the input window into a file,

Batch Run this will open a browser window permitting the input of a file name which is immediately executed. The output is written to the `...log` and `...lst` files.

Exit for exiting the application.

You can either type CMAT input into the input window, or you can cut and past using `CTRL x`, `CTRL c`, and `CTRL v` keys or you can load a script using the corresponding button. In addition the following keys can be used for moving the cursor and view in the display:

Page Up, Page Down for moving cursor and view one page up or down in the output.

Home, End for moving the cursor to the begin or end of the current line.

CTRL Home, CTRL End for moving the cursor to the first or last line of the output window.

To run CMAT interactively with command line input and standard output (e.g. by using NTEmacs with BASH oder some kind of DOS extender) you need to invoke CMAT by typing `cmat` into the command line and start the program input after an opening compound statement { on a new input line. CMAT is terminated either with an `exit;` statement or by closing the initial compound statement with }. Therefore, the shortest CMAT program is `cmat` { ; } . When passing through the input statements the first time, the statements are compiled and executed simultaneously, with the following exceptions, where execution but not the compilation is postponed:

forward jump : This is a `goto` statement with a target label later in the code. The execution is delayed until the target label is found.

if statement : The execution is delayed until the next statement is started. If the `if` statement is connected with a compound statement, the execution is performed until the compilation of the first statement after the closing brace } is started.

switch statement : The execution is delayed until the first statement after the related compound statement is found.

for and while statement : This is equivalent to the `if` statement.

conditional operator : The execution of the `?` and `:` conditional statement is delayed until the next statement is started.

In later passes the compiled assembler code is executed. By default the output is directed to standard output but can be redirected using the `logfile(`*filename*`)` or `txtfile(`*filename*`)` standard functions. Calling the two functions without an input argument *filename* redirects the output to standard output. During the program input, an `#include "`*pathname*`"` can redirect the standard input to the input file *pathname*. This feature specifically permits the execution of CMAT commands in an ASCI file which was generated during the run. After executing the commands in the included input file, the program input automatically returns to the standard input or the file from where the last `#include` command was issued. Note, that the length of the *pathname* argument is restricted (see page 21).

Online help is available only when working with the Frontend. Otherwise the corresponding .pdf files in `cmat\doc` must be opened by the user. The document files make use of the *hyperref* package which permits bookmarks.

3.2.2 Execute CMAT in Batch Mode

You may use the Tcl TK frontend which is delivered with CMAT, however you can also execute CMAT statements inside an ASCII file *fname.inp* by typing `cmat fname`. The input file *fname.inp* must then start with an opening { brace of a compound statement. Like in interactive mode, `#include "`*pathname*`"` commands redirect the program input to other files specified in the pathname. This feature makes it possible to execute CMAT code which was written to an ASCI file earlier at the same run. By default all output is directed to logfile *fname.log* and the output file *fname.txt*. If there exist already a file with the same name *fname.log* (resp. *fname.txt*) the old files are renamed into *fname.blg* (resp. *fname.bxt*) to prevent the loss of old results. However, like in interactive mode, the output can be redirected to other files or standard output using the `logfile()` and `txtfile()` functions.

3.2.3 Utility Files

During it's execution CMAT generates a number of utility files. To enable simultaneous batch runs of `cmat` the file names are extended by an integer number which is the job process ID *pid*.

__opro__... created with every execution of `cmat`

__memb__... created with every execution of `cmat`

__data__... created with every execution of `cmat`

__scra__... created with every execution of `cmat`

_uf_... prefix of utility files created by specific functions.

These are temporary files which normally are deleted during a normal exit of `cmat`, e.g. using the closing }. However, as a result of aborted runs a number of those files may remain in your working directory, e.g. `mytest`. Those are being deleted at the begin of the next execution of `cmat`. They could also be deleted by running the batch file `uf_kill`. The frontend has a specific command for that in the `File` pulldown menu.

Chapter 4

Restrictions and Comparisons

4.1 Some Shortcomings

Due to the limited time and financial resources of the author we have to report the following list of shortcomings in CMAT:

1. Other Operating Systems: At an early stage, CMAT was developed in UNIX using BSD Lex and Yacc. A host.h header file still defines most of the differences, i.e. path names, functions with variable arguments, DLL declarations. Therefore, it should be quite easy to make running versions for Unix, Linux, and the Mac.
2. Dynamic Binding: It should be possible to link object code of subroutines written in C, C++, or Fortran to the executable. Specifically for the Windows OS, the linking of user defined subroutine DLL's should be possible.
3. String processing: Not many operations with string arguments are available at this time. However, most of the string processing subroutines of the standard C runtime library are available.
4. Arrays with more than 2 dimensions (tensors) are now supported but some tensor operations, especially with tensors containing string data, may still have some problems.
5. Not all index operations try to preserve prior assigned row and column names.

4.2 Length Restrictions

The following are constants, which may be easily changed for a more customary version of CMAT:

- Maximum length of variable, function, or file names is 32 chars.
- Maximum length of row and column names is 32 chars.
- Maximum length of row and column labels is 80 chars.
- Maximum path length of directories (used in the arguments of some functions) is 256 characters.

- Maximum length of an input token is 1024 characters. Note, that this normally restricts the length of input string data (embedded into quotes). There is NO restriction other than the available memory for the internal length of string data which must be provided to Yacc and Lex. Note, that the length comments is not restricted.
- A maximum of 10 lookup directories may be specified.

4.3 Differences with C Language

Most of the syntax of CMAT is compatible to that of the C language when working with scalar data types. However, there are a few exceptions.

1. Differences when working with scalars:
   - The scope keywords `extern` and `static` are not implemented. Only inside functions are variables locally defined. However, variables can be easily reset, freed, or reallocated, e.g. when using type declarations or simple assignments.
   - In CMAT variable types do not need to be declared, but variables can be casted to specific types. If type declarations are used then the declared variables are reallocated and initialized. A variable should be casted if only its type but not its value should be changed.
   - CMAT has only one integer type (which is always long int) and only one floating point data type referred to as `real`, which is internally double precision.
   - CMAT permits the complex data type and easier ways of string processing.
   - CMAT provides additional casting like `complex`, `conj`, and `imag`.
   - There is obviously no need for pointers in CMAT.
   - Subroutine definitions can appear anywhere in the code but must be defined before being called.
   - Input and returned arguments in subroutines are separated in CMAT. For multiple returns, the return statement in the function definition must have a list of variable names in parentheses, e.g. `return(a,b,c);`. The function can then be called in the form `<a,b,c> = f(d,e,f);`, where arguments `d,e,f` are inputs and arguments `a,b,c` are returned.
   - Function calls may contain a `global` clause. The `global` clause could be necessary when functions are called by other functions and must have a specific number of input arguments.
   - The CMAT language provides the additional `link`, `option`, and `poptn` keywords.
   - In CMAT many of the standard functions have default argument settings and some of the standard functions return different types of results.
   - Some of the standard functions were not implemented because there was either no need (e.g. memory allocation) or they did not seem to be important enough in an early release.
2. Differences when working with vectors, matrices, and tensors:
   - Data objects in CMAT can be scalars, vectors, matrices, tensors, kD trees, and single indexed lists of those objects. Data lists may contain the following data objects:
     - numerical (int, real, complex) and string scalars
     - vectors and matrices
     - tensors

  - (sub-)lists

- In CMAT matrix entries $a_{i,j}$ are referred to in the form `a[i,j]` and not like `a[i][j]` as in C.
- A tensor `t[i,j,...]` is indexed in the same way.
- Tensors, matrices and vectors may be initialized in type declarations, e.g. `real a[5,15]=1.;` is a valid initialization of the 5×15 matrix $\mathbf{A}$.
- Lists are initialized in the form `list l[5];` specifying the name and an initial length of the list. Each entry then is initialed to a scalar with missing value.
- Structs are declared using the keyword `struct`. Entries of structs are referred to by compound names, where the struct name is separated from the entry name by a dot. E.g. `a.b.c` means that struct `a` has a struct entry `b`, and substruct `b` has an enyry `c`.
- The additional `free` keyword can be used to free memory of strings, vectors, matrices, tensors, kD trees, or entire lists which are no longer needed in the code.
- CMAT provides additional casting like `herm`, `symm`, and `tri2sym` to matrix objects.

Chapter 5

Tutorial: Basic Elements of the CMAT Language

5.1 Defining Scalars

CMAT works with four basic types of data: `int`, `real`, `complex`, and `string` (including *char* as special case of strings with one character length). One variables data type is automatically defined from the type of operation which creates its values. A scalar variable can be defined by a simple assignment statement:

Data Type	Internal Rep.	Example
`int` variable a	long integer	`a = -100;`
`real` variable b	double float	`b = 1.14e-5;`
`complex` variable c	two double float	`c = 5.5 - 5i;`
`string` variable e	for length see page 21	`e = "Hello, World";`

Data Types

Remarks:

- In C (see [455]), string data are embedded in double quotes and are processed as arrays of one character data which are embedded into single quotes (e.g. defined by `char a; a = 'c';`). Like in C, a string can contain a new line character only if it is proceeded by a backslash (\) character. To simplify applications, in CMAT char data are just a special case (length 1) of string data and there is practically no difference between the use of single and double quotes.

- You can assign an `int` number to a `real` variable simply by using a decimal point, for example `f = -100.;` defines a `real` variable f.

- An important special value of a `real` variable is a *missing value* which is denoted by a single period. Many of the functions will return missing values when called with invalid arguments. For example, calling the `sqrt` function with a `string` argument results in a missing value. Also, you are highly recommended to test the return value of a function: the input `r = sqrt("bad arg"); if (r == .) then exit;` terminates the execution of program statements.

- The real and imaginary part of a complex number is separated by a `+` or `-` sign. If the real part is zero you can use *number*`i` for an imaginary number. It is however important that the imaginary unit i on the end is preceded by a number **without white space** between number and i. For example, `d=5-1i;` is a complex number, but `d=5-i;` is an arithmetic expression for which the variable i must be set before this statement can be executed.
- For a scalar variable, you can separate the real and imaginary part of a complex number with a white space as in `c = 5.5 - .5i;`, where the - defines the subtraction between a real and a imaginary number. When defining vectors, matrices, or tensors where the number of entries is significant, you must write a complex number in compact form **without** a white space between real and imaginary part, `c = 5.5-.5i;`, avoiding the confusion with two separate numbers (one real and one imaginary).
- Like in C (see [455], p. 42) you may cast variables to specific data types like `int` or `real`. For example, you can use `(int)a % (int)b` to convert the floating point values of the variables a and b into integers by chopping off the decimal part. Or you may use `(real)i * (real)j` to prevent the integer overflow of an arithmetic operation.
- Note, that CMAT treats the `int` specification as long internally, the `real` specification as double data type, and the `complex` type as two double precision floating point numbers.
- An identifier (name of a variable or function) must start with a letter and can be followed by a restricted number (see page 21) of significant letters, digits, or underline symbols.
- Like in C, comments must be embedded inside `/*` and `*/` brackets, can extend across line breaks, and may be included at ANY location of a CMAT program. There is no restriction on the length of comments.

In the following statements the type of the variable a is changed from `int` to `complex` and then redefined to `string`:

```
a = -2; print " INT: a=", a;
a = sqrt(a); print " COMPLEX: a=", a;
a = " Now a is a string"; print " STRING: a=", a;
```

producing the following printed output (in interactive mode):

```
[2]   a = -2; print " INT: a=", a;

 INT: a=-2
[3]   a = sqrt(a); print " COMPLEX: a=", a;

 COMPLEX: a= 0.0000 + 1.4142i
[4]   a = " Now a is a string"; print " STRING: a=", a;

 STRING: a= Now a is a string
```

5.1.1 Use Hexadecimal Constants

The language (Yacc and Lex script) was extended for the input of hexadecimal numbers. They are converted into long integers and therefore must not be too large. Hexadecimal constants must begin with `0x` and may contain the digits $0, \ldots, 9, a, b, c, d, e, f$.

```
h1 = 0xabc1234;
print "Hexadecimal(0xabc1234) =",h1;
h2 = 0x2fddddd3;
print "Hexadecimal(0x2fddddd3)=",h2;
```

```
Hexadecimal(0xabc1234) = 180097588
Hexadecimal(0x2fddddd3)= 803069395
```

We found hexadecimal constants especially important for setting some specific seeds for uniform random generators:

1. Single source base 256 integer generator with single 256 digit deck (Richardson, 2011)

```
print "ISS1: Seed default: 0xabc1234: hexadecimal";
srand(0xabc1234,"iss1");
mu3 = rand(nr,1,'g',"iss1");
print "ISS1=", mu3;
```

```
ISS1: Seed default: 0xabc1234: hexadecimal

ISS1=
   |             1
------------------
 1 |     982600291
 2 |     735314717
 3 |    4087362757
 4 |    1802693449
 5 |    3799537722
 6 |     135030984
 7 |    3729099871
 8 |    2704712113
 9 |     349760511
10 |    2529659426
```

2. Twin source base 256 integer generator with four 65536 digit decks (Richardson, 2011)

```
print "ITS4: Seed default: 0x2fddddd3: hexadecimal";
srand(0x2fddddd3,"its4");
mu4 = rand(nr,1,'g',"its4");
print "ITS4=", mu4;
```

```
ITS4: Seed default: 0x2fddddd3: hexadecimal

ITS4=
```

```
   |            1
---+---------------
 1 |    3647235203
 2 |    1095945229
 3 |    3446603839
 4 |    1324252004
 5 |    3128063493
 6 |     638329403
 7 |    1752226044
 8 |    2687002194
 9 |     895754293
10 |    3862544476
```

5.2 Some Terms Used in Matrix Algebra

An $m \times n$ matrix $\mathbf{A}$ is a matrix consisting of values a_{ij} in m rows $i = 1 \ldots, m$ and n columns $j = 1, \ldots, n$,

$$\mathbf{A} = (a_{ij}) = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}.$$

If the $m \times n$ matrix $\mathbf{A}$ contains complex elements $a_{ij} \in \mathcal{C}$ then it belongs to the complex vector space $\mathcal{C}^{m \times n}$, i.e. $\mathbf{A} \in \mathcal{C}^{m \times n}$, otherwise it belongs to the real vector space $\mathcal{R}^{m \times n}$, i.e. $\mathbf{A} \in \mathcal{R}^{m \times n}$.

A matrix with $m = 1$ row and $n > 1$ columns is called *row vector*, a matrix with $m > 1$ rows and $n = 1$ columns is called *column vector*. A matrix which has the same number of rows as columns is called *square matrix*. The elements a_{ii} form the *diagonal* of matrix $\mathbf{A}$. A matrix which contains zero elements in all offdiagonal locations is called *diagonal matrix*, i.e. $a_{ij} = 0$ for all $i \neq j$. A matrix which contains nonzero elements only on and below of its diagonal is called *lower triangular matrix*, i.e. $a_{ij} = 0$ for all $j > i$. A matrix which contains nonzero elements only on and above of its diagonal is called *upper triangular matrix*, i.e. $a_{ij} = 0$ for all $j < i$. If a matrix contains only a few nonzero diagonals beneath or above its main diagonal it is called *band diagonal*. The matrix $\mathbf{A} = (a_{ij})$ has *upper bandwidth* q if $a_{ij} = 0$ for all $j > i + q$ and $\mathbf{A}$ has *lower bandwidth* p if $a_{ij} = 0$ for all $i > j + p$. Specifically, a matrix is called *tridiagonal* if it has lower and upper bandwidth equal to one, i.e. there is $a_{ij} = 0$ for all $j > i + 1$ and all $i > j + 1$. A matrix with full upper triangle and a lower bandwidth of one is called *upper Hessenberg* matrix; a matrix with full lower triangle and a upper bandwidth of one is called *lower Hessenberg* matrix.

By exchanging rows with columns of a $m \times n$ matrix $\mathbf{A} = (a_{ij})$ we obtain the *transposed* $n \times m$ matrix $\mathbf{A}^T = \mathbf{B} = (b_{ij})$ with $b_{ij} = a_{ji}$. The *conjugate transposed* matrix $\mathbf{B} = \mathbf{A}^H$ of a $m \times n$ complex matrix $\mathbf{A}$ is defined by $b_{ij} = \overline{a}_{ji}$.

A square $n \times n$ matrix $\mathbf{A}$ is called *symmetric matrix* if the transposed matrix $\mathbf{A}^T$ equals $\mathbf{A}$, i.e. if $a_{ij} = a_{ji}$ for all its entries $i, j = 1, \ldots, n$. A $n \times n$ complex matrix is called *Hermitian matrix* if the conjugate transpose $\mathbf{A}^H$ equals $\mathbf{A}$, i.e. $a_{ij} = \overline{a}_{ji}$. (Necessarily, a Hermitian matrix must contain only real diagonal elements.)

A matrix containing only zero entries is called *null matrix*. The square $n \times n$ matrix $\mathbf{A}$ that contains unit diagonal elements $a_{ii} = 1$ for all $i = 1, \ldots, n$ and zero off-diagonal entries $a_{ij} = 0$ for $i \neq j$ is called *identity matrix* and is usually denoted by $\mathbf{I}$ or $\mathbf{I}_n$.

An *orthogonal matrix* $\mathbf{A}$ is a real square matrix which when multiplied with it's transpose results in the identity matrix, $\mathbf{A}^T\mathbf{A} = \mathbf{A}\mathbf{A}^T = \mathbf{I}$. A real $m \times n$ matrix $\mathbf{A}$ with

- $\mathbf{A}^T\mathbf{A} = \mathbf{I}_n$ is *column orthogonal*,
- $\mathbf{A}\mathbf{A}^T = \mathbf{I}_m$ is *row orthogonal*.

An *unitary matrix* $\mathbf{A}$ is a complex square matrix which when multiplied with it's complex conjugate transpose results in the identity matrix, $\mathbf{A}^H\mathbf{A} = \mathbf{A}\mathbf{A}^H = \mathbf{I}$.

5.3 Defining Vectors and Matrices

If there is no need to distinguish between vectors and matrices, this text simply refers to matrices since vectors can be considered as matrices with one row or one column. There are different ways to create and define matrices with CMAT. Of course when the right-hand side of an assignment results in a matrix, the variable on the left-hand side is defined as a matrix and can be used in following statements as long as the variable name is not newly defined. Additionally, there are some more ways to define a matrix:

1. Using type declarations for matrix initializations.
2. Using brackets `[` and `]`: constants, scalars, or compatible matrices can be listed inside `[` and `]` brackets defining a matrix. For example,
3. Using functions like `cons`, `rand`, `ide`, `randperm`.
4. Using concatenation operators: constants, scalars, or compatible matrices can be concatenated horizontally, vertically, or diagonally.
5. Using indexed assignments: variable names on the left-hand side of assignment statements can be followed by indices in brackets `[` and `]` defining the location where the first element of the object on the right-hand side of the assignment is attached.

5.3.1 Matrix Definition by Type Declaration

The type declarations `int`, `real`, `complex` and `char` can be used to allocate memory for (column) vectors, rectangular dense matrices, and tensors with constant values. If the declaration does not contain an explicit initialization by a constant or a scalar identifier, the entries are initialized to zero.

Using the `int` type declaration a vector a matrix with (long) integer values can be allocated and initialised:

```
int ind[5] = 1;
print ind;
```

```
   |   1
--------
 1 |   1
 2 |   1
 3 |   1
 4 |   1
 5 |   1
```

The following type declaration creates a 3×3 symmetric matrix with double precision floating point values:

```
real dbl[3,3] = 5.5;
print dbl;
```

```
 S |         1         2         3
-----------------------------------
 1 |   5.50000
 2 |   5.50000   5.50000
 3 |   5.50000   5.50000   5.50000
```

You may also use an initialized scalar variable for setting vector and matrix entries:

```
z = 1.+5.i;
complex cmp1[5] = 1.+5.i,
        cmp2[3] = z;
print cmp1, cmp2;
```

```
   |                    1
-------------------------
 1 |   1.00000 + 5.00000i
 2 |   1.00000 + 5.00000i
 3 |   1.00000 + 5.00000i
 4 |   1.00000 + 5.00000i
 5 |   1.00000 + 5.00000i

   |                    1
-------------------------
 1 |   1.00000 + 5.00000i
 2 |   1.00000 + 5.00000i
 3 |   1.00000 + 5.00000i
```

5.3.2 Matrix Definition Inside Brackets (Literal)

Define the matrix by listing its elements. For example, the following notation specifies a 2×2 lower triangular matrix **A**,

```
  a = [ 15 ,
        22  51];
print a;
```

```
 L |    1    2
--------------
 1 |   15
 2 |   22   51
```

The matrix is defined rowwise inside the brackets, a comma separator indicates continuation on a new row. The maximum of elements across all rows defines the number of matrix columns. Since CMAT is similar to C, it's matrix operations are rowwise oriented. Some FORTRAN based matrix programming languages are columnwise oriented. We will denothe this kind of matrix definition as *matrix literal* similar to its use in SAS/IML®.

Using complex numbers in matrix definitions requires you to write the whole complex number without using white space between the real and imaginary part:

```
c = [ 5.3+0.2i 6.1-2.5i,
      .       7.+2.22i];
print c;
```

Here the `print` command is used to print the matrix **C** in a default format (later you will be told how to modify the default output of the `print` statement),

```
|                    1                    2
```

```
-------------------------------------------
 1 |   5.30000 + 0.20000i   6.10000 - 2.50000i
 2 |    .      +  .     i   7.00000 + 2.22000i
```

The element $c_{2,1}$ is a so-called *missing value* and is written as a period. There is some limited missing value algebra provided in CMAT. Missing values are used mostly for indices (specifying all indices of one dimension) or missing arguments of standard functions. A missing value is not a zero, therefore CMAT does not assume that **C** is an upper triangular matrix.

You may also list the names of variables or matrices inside the brackets as long as they already have values assigned in former statements. For example, using the former specification of matrix **A**, the following statement also specifies a lower triangular matrix **L**,

```
aa = [ a ,
       a  a];
print aa;
```

CMAT is smart enough to determine that **L** is lower triangular and prints the following matrix `aa`:

```
 L |    1    2    3    4
-----------------------
 1 |   15
 2 |   22   51
 3 |   15    0   15
 4 |   22   51   22   51
```

By default, the upper left corner of the output tells the type of the matrix using one of the following abbreviations [1]:

Abbreviation		Type
D	DIA	diagonal
S	SYM	symmetric
L	LOW	lower triangular
U	UPP	upper triangular
O	ORT	orthogonal

This form of matrix literal has the advantage that it may contain the names of other objects and so can be used to describe supermatrices by means of submatrices. However, his form has the disadvantage when dealing with a large number of string data. Therefore, the grammar of CMAT was extended to permit a second form of matrix literal useful for specifying matrices with many string data. In that, the enclosing bracket tokens `[` and `]` are replaced by tokens `["` and `"]` where no white space is permitted between the two symbols, bracket and quote. To illustrate this, the following two examples result in the same mixed data type matrix. This is the common form of matrix literal, which would also permit to use object names as entries:

```
school = [ 1 "regular"   "self"  10 ,  1 "regular"   "team"  17 ,
           1 "regular"   "class" 26 ,  1 "afternoon" "self"   5 ,
```

[1] This output does not reveal the form used to store the object, e.g. dense, packed, or sparse. The `attrib()` function can be used to obtain information about the storage type.

```
            1 "afternoon" "team"  12 ,  1 "afternoon" "class" 50 ,
            2 "regular"   "self"  21 ,  2 "regular"   "team"  17 ,
            2 "regular"   "class" 26 ,  2 "afternoon" "self"  16 ,
            2 "afternoon" "team"  12 ,  2 "afternoon" "class" 36 ,
            3 "regular"   "self"  15 ,  3 "regular"   "team"  15 ,
            3 "regular"   "class" 16 ,  3 "afternoon" "self"  12 ,
            3 "afternoon" "team"  12 ,  3 "afternoon" "class" 20 ];


cnam = [ "School" "Program" "Style" "Count" ];
school = cname(school,cnam);
print "Data 1=", school;
```

The new form does not need the so many quotes around each string for telling CMAT that the identifiers are strings. Please note, that there must not be white space separating the brackets and strings, the two symbols are recognized as one token.

```
school2 = [" 1 regular   self  10 ,  1 regular   team  17 ,
             1 regular   class 26 ,  1 afternoon self   5 ,
             1 afternoon team  12 ,  1 afternoon class 50 ,
             2 regular   self  21 ,  2 regular   team  17 ,
             2 regular   class 26 ,  2 afternoon self  16 ,
             2 afternoon team  12 ,  2 afternoon class 36 ,
             3 regular   self  15 ,  3 regular   team  15 ,
             3 regular   class 16 ,  3 afternoon self  12 ,
             3 afternoon team  12 ,  3 afternoon class 20 "];
school2 = cname(school2,cnam);
print "Data 2=", school2;
```

Of course this form permits no object names as entries, since each identifier (name starting with character) is defined as a string.

There are two problems to watch now:

- The `["` and the `[ "` tokens have now different meanings. For example: `a = [ "x1":"x6" ];` and `b = ["x1":"x6"];` create the two different vectors:

```
a=
   |    1    2    3    4    5    6
-----------------------------------
 1 |   x1   x2   x3   x4   x5   x6


b=
   |    1    2    3
------------------
 1 |   x1    :   x6
```

Maybe quoted strings inside the `["` and `"]` tokens should not be permitted.

- Note the difference between `s1=[ "old way" ];` and `s2=[" new way "];`

```
s1= old way

s2=
   |     1     2
---------------
 1 |   new   way
```

Abbreviations of Matrix Types in `print` Output

Note, that the objects inside brackets are rowwise concatenated and therefore the elements of each row must have the same row number. The number of columns may differ. They are filled with zeros to equally maximum length. You cannot use function calls or expressions inside the bracketed matrix definition. However, you can use the following two operators inside the bracket definition:

- *Use of colon operator:*
  The colon operator `a:b:c` when applied to numeric constants a, b, and c generates a series of values

$$a, a+b, \ldots, a+k*b \quad , \quad \text{where} \quad \begin{cases} a+k*b \le c & \text{for} \quad b>0 \\ a+k*b \ge c & \text{for} \quad b<0 \end{cases}$$

  with a nonnegative integer k. The short form of the colon operator, `a:c`, implicitly assumes `b=1`. Note, that only constants and identifiers (names of scalar variables) can be used with the colon operator, i.e. you cannot use a construction like `[n+1:n+p]`. For example, the statement

```
a = [ -1:5 -7:2:0,
      -4:2 -5:2 ];
```

  generates the following 2×15 matrix:

```
    |   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
-------------------------------------------------------------------
 1 |  -1   0   1   2   3   4   5  -7  -5  -3  -1   0   0   0   0
 2 |  -4  -3  -2  -1   0   1   2  -5  -4  -3  -2  -1   0   1   2
```

  The colon operator can also be applied on the numeric suffix of string constants:

```
s = [ "a2":2:"a8"  "b3":1,
      "c4":2       "d4":2];
print s;
```

  Since the first row is specified with seven entries and the second row only with six entries, the second row is filled with a blank string:

```
    |    1    2    3    4    5    6    7
-------------------------------------
 1 |   a2   a4   a6   a8   b3   b2   b1
 2 |   c4   c3   c2   d4   d3   d2
```

- *Use of pound operator:*
  The pound operator `a#b` copies a times the scalar value of b, where a must be a positive integer. For example, the following matrix

```
bb = [ 2#"a"  3#12.5,
       3#"b2"   "d2"];
print bb;
```

  contains 5 entries in its first row, and therefore, the second row is filled up with one zero,

```
   |         1         2         3         4         5
---------------------------------------------------------
 1 |         a         a   12.5000   12.5000   12.5000
 2 |        b2        b2        b2        d2    0.0000
```

Similarily, a row vector of strings with numerical suffix is generated by

```
s = "string1";                |          1          2          3          4          5
ss = [ s : 2 : 10];          ------------------------------------------------------------
print ss;                     1 |    string1    string3    string5    string7    string9
```

Note that the string prefix is taken only from the first element of the colon construct. You can, however, write the same or any other `string` prefix with the last element. CMAT only uses the **numeric** suffix of the last element of the colon construct. The bracket definition of a matrix or vector may be used in arithmetic expressions or for arguments of function calls.

5.3.3 Matrix Definition by Functions

The following functions can be used for defining matrices: `ide(`n`)`, `cons(`$nr, nc, val,\ type$`)`, `rand(`$nr, nc, type, \ldots$`)`, `spmat(`$nr, nc, rind, cind, val <, sopt >$`)`, `mrand(`$kind, a, b$`)`, and `randperm(`$n$`)`. The functions `shape(a<,nrb><,ncb>)`, `replace(a,old,new<,rel>)`, `diag(a<,k>)`, and `dia2vec(a<,k>)` are useful for basic modifications on matrices.

Function ide(n)

The function `ide(`n`)` returns an $n \times n$ identity matrix, where n is the only integer input argument. For example,

```
print ide(3);
```

generates the 3×3 identity matrix

```
 D |         1         2         3
-----------------------------------
 1 |   1.00000
 2 |             1.00000
 3 |                       1.00000
```

Function cons(nr,nc,val,mtyp)

You can create matrices that have the same (constant) value for each element using the `cons()` function. This function may have a maximum of four arguments. Each of the four arguments is optional, however either nr or

nc must be specified. The first two arguments *nr* and *nc* must be of type `int`, specifying the number of rows, *nr*, and the number of columns, *nc*. When specifying a missing value (period) for either *nr* or *nc*, a default value is chosen depending on the matrix type (see below). The third argument specifies a `int`, `real`, or `complex` scalar value *val* defining the constant value of the matrix. By default, $val = 0$. For example, in

```
c1 = cons(5);
c2 = cons(2,5);
c3 = cons(2,5,1);
print c1, c2, c3;
```

the first command produces a column vector with 5 rows of zeros, the other two commands produce 2×5 matrices containing zeros and ones, where CMAT decides to store `c2` as a sparse 2×5 diagonal matrix:

```
    |   1
--------
 1  |   0
 2  |   0
 3  |   0
 4  |   0
 5  |   0
```

```
 D  |   1   2   3   4   5
-----------------------
 1  |   0
 2  |       0
```

```
    |   1   2   3   4   5
-----------------------
 1  |   1   1   1   1   1
 2  |   1   1   1   1   1
```

The fourth argument *mtyp* is of type `char` specifying the type of the matrix.

mtyp	**Description**
'g'	Creates $nr \times nc$ full rectangular matrix. This is the same as specifying less than four arguments or specifying a missing value.
'd'	Creates a $nr \times nr$ diagonal matrix. If one of the first two arguments is not specified it is set equal to the other argument.
'u'	Creates a $nr \times nr$ upper triangular matrix. If one of the first two arguments is not specified it is set equal to the other argument.
'l'	Creates a $nr \times nr$ lower triangular matrix. If one of the first two arguments is not specified it is set equal to the other argument.

Specification of Matrix Types for `cons` Function

If the second argument *nc* is not specified, then $nc = 1$ for *mtyp*='g' and $nc = nr$ for *mtyp*= 'l', 'u', or 'd'. For example, the following commands

```
c4 = cons(3,3,"diag",'d');
c5 = cons(3,.,"lower",'l');
print c4,c5;
```

produce the following output

```
 D  |      1      2      3
-------------------------
 1  |   diag
 2  |          diag
 3  |                 diag
```

```
 L  |       1       2       3
-----------------------------
 1  |   lower
 2  |   lower   lower
 3  |   lower   lower   lower
```

Function spmat(nr,nc,rind,cind,val<,sopt>)

Function `spmat` can be used to create a sparse matrix by specifying the nonzero index locations and values. The arguments `nr` and `nc` specify the number of rows and columns of the sparse matrix `A`. The N values of the vector `val` specify the nonzero entries of the sparse matrix **A**. Depending on the setting of the string option `sopt="coord", "rowor", "color", "rowsym", "colsym"` different input types are possible:

coord The matrix is defined in coordinate form, that means, the three arguments `rind,cind,val` must be vectors of equal size specifying the nonzero row and column locations and corresponding values. This is the default.

rowor The matrix is defined in row-oriented form, that means, the `rind` argument is a vector with nr entries specifying the upper index limit of the nonzero values in each row w.r.t. to the N vectors `cind` and `val`. That means the value of `rind[nr]` must be equal N.

color The matrix is defined in column-oriented form, that means, the `cind` argument is a vector with nc entries specifying the upper index limit of the nonzero values in each column w.r.t. to the N vectors `rind` and `val`. That means the value of `cind[nc]` must be equal N.

rowsym The matrix `A` is symmetric and is defined in row-oriented form like `rowor`, however, only one entry of each symmetric pair must be specified.

colsym The matrix `A` is symmetric and is defined in column-oriented form like `color`, however, only one entry of each symmetric pair must be specified.

Note, that duplicate index settings are removed as well as zero values in argument `val`. The following input creates 3 matrices.

```
nr = nc = 5;
rind= [ 1 2 3 5 7 ];
cind= [ 1  2  3  1 4  2 5];
vals= [ 5. 5. 5. 1. 5. 2. 5. ];
a = spmat(nr,nc,rind,cind,vals,"rowor");
c = spmat(nr,nc,rind,cind,vals,"rowsy");
print a, c;
rind = [ 1 2 3 4 4 5 5 ];
b = spmat(nr,nc,rind,cind,vals,"coord");
print b;
```

Matrices `a` and `b` are lower triangular and matrix `c` is symmetric:

```
 L |         1         2         3         4         5
-------------------------------------------------------
 1 |   5.00000
 2 |   0.00000   5.00000
 3 |   0.00000   0.00000   5.00000
 4 |   1.00000   0.00000   0.00000   5.00000
 5 |   0.00000   2.00000   0.00000   0.00000   5.00000
```

```
 S |         1         2         3         4         5
---------------------------------------------------------
 1 |   5.00000
 2 |   0.00000   5.00000
 3 |   0.00000   0.00000   5.00000
 4 |   1.00000   0.00000   0.00000   5.00000
 5 |   0.00000   2.00000   0.00000   0.00000   5.00000

 L |         1         2         3         4         5
---------------------------------------------------------
 1 |   5.00000
 2 |   0.00000   5.00000
 3 |   0.00000   0.00000   5.00000
 4 |   1.00000   0.00000   0.00000   5.00000
 5 |   0.00000   2.00000   0.00000   0.00000   5.00000
```

Note, that the option string `"coord"` could be dropped for the same result, since this is the default.

Functions srand(seed), rand(nr,nc,'mtyp',"dist",...), and mrand("kind",a,b)

Function `rand()` may have a maximum of seven arguments, however, all arguments of `rand` are optional. If you do not specify any argument, `rand()` is compatible with the C language function and returns an `int` scalar value i calling the pseudo-random generator of the C compiler which was used in the compilation of CMAT. The pseudo-random generator can be initialized using the C function `srand(seed)`, with positive `seed`. If `rand()` is called without prior initialization by `srand(seed)`, the series of values returned by `rand` is the same as initialized by `srand(1)` (see [354]). The integer result is machine dependent:

1. In Domain C using BSD4.3, the result is in the range from 0 to $2^{31} - 1$. It can be scaled into $[0, 1]$ with division by 2147483647, which is the largest signed long integer. The first call of `rand();` produces `1103527590`.
2. The PC version (Windows NT) of `rand()` generates values inbetween 0 and $2^{15} - 1 = 32767$, i.e. can be scaled into $[0, 1]$ with division by 32767. The first call produces `16838`.

The `rand` function was extended to more arguments. The first three arguments define the number of rows, *nr*, number of columns, *nc*, and the matrix type, *mtyp*, similar to corresponding arguments of `cons`. All the *mtyp* specifications of `cons` are available in `rand`, and some additional specifications are permitted. Depending on the value of *mtyp*, more arguments can be specified for the `rand` function. Table 5.3.3 on page 39 illustrates the syntax for the `rand` function determined by the *mtyp* specification. The fourth argument with *mtyp* = `'g', 'd', 'u', 'l'`, or `'s'` is a string argument listed in double quotes specifying one of the distribution forms shown in table 5.3.3 on page 40.

The basis for all these distributional pseudo random numbers is the random number generator developed by Fishman & Moore [258]. If no more than three arguments are specified, the default distribution is univariate in $[0, 1]$. For illustration, the following three calls of `rand`

```
print rand(3), rand(3,2), rand(3,.,'d');
```

produce the following output: a column vector, a 3×2, and a 3×3 diagonal matrix,

```
   |         1          |         1         2         D |         1         2         3
-------------         -----------------------        -----------------------------------
 1 |   0.18496         1 |   0.25940   0.92160         1 |   0.06657
 2 |   0.97009         2 |   0.96928   0.54298         2 |             0.81932
 3 |   0.39982         3 |   0.53169   0.04979         3 |                       0.52387
```

The first two of the following statements generate a general and a diagonal matrix of univariate distributed random numbers in $[4, 10]$ and $[2, 10]$. The third statement generates a symmetric matrix of rank 2 which has one zero eigenvalue. Due to rounding errors of the eigenvalue function, it's value is about 1.e-17, i.e. not exactly zero in machine precision.

```
r1 = rand(3,2,.,.,4,10);
r2 = rand(3,.,'d',.,2,10);
r3 = rand(3,.,'e',2);
print r1, r2, r3;
print eig(r3);
```

produce the output (note, that the upper left corner of the output contains an abbreviation for matrices with a specific form, like diagonal, symmetric, lower and upper triangular matrices)

```
   |         1         2              D |         1         2         3
-----------------------              -----------------------------------
 1 |   9.12037   4.40311              1 |   9.81412
 2 |   9.74214   5.78316              2 |             3.81206
 3 |   5.63567   8.13958              3 |                       7.50589

 S |         1         2         3      D |         1         2         3
---------------------------------      -----------------------------------
 1 |   0.01788                          1 |   -5e-017
 2 |  -0.01455   0.42662                2 |             0.12118
 3 |   0.04533  -0.06173   0.11638      3 |                       0.43970
```

and the following statements illustrate that a square column orthogonal matrix is also row orthogonal

```
r4 = rand(3,3,'o');                  O |         1         2         3
rr1 = r4' * r4; rr2 = r4 * r4';     -----------------------------------
print r4, rr1, rr2;                   1 |  -0.28052  -0.57315   0.76993
                                      2 |  -0.64206   0.70831   0.29335
                                      3 |  -0.71349  -0.41205  -0.56670

 S |         1         2         3      S |         1         2         3
---------------------------------      -----------------------------------
 1 |   1.00000                          1 |   1.00000
 2 |   -6e-017   1.00000                2 |   -2e-016   1.00000
 3 |    6e-017   -8e-017   1.00000      3 |   0.00000   -8e-017   1.00000
```

mtyp	Syntax	Description
'g'	rand(nr,nc,'g',"dist",...)	Creates $nr \times nc$ full rectangular matrix of random numbers with the specified distribution. The last arguments may specify distributional parameters for which however default settings exist. This is the same as specifying less than four arguments or specifying a missing value.
'd'	rand(n,n,'d',"dist",...)	Creates a $n \times n$ diagonal matrix of random numbers with the specified distribution. The last arguments may specify distributional parameters for which however default settings exist. If the second argument is specified it must be equal to the first.
'u'	rand(n,n,'u',"dist",...)	Creates a $n \times n$ upper triangular matrix of random numbers with the specified distribution. The last arguments may specify distributional parameters for which however default settings exist. If the second argument is specified it must be equal to the first.
'l'	rand(n,n,'l',"dist",...)	Creates a $n \times n$ lower triangular matrix of random numbers with the specified distribution. The last arguments may specify distributional parameters for which however default settings exist. If the second argument is specified it must be equal to the first.
's'	rand(n,n,'s',"dist",...)	Creates a $n \times n$ symmetric matrix of random numbers with the specified distribution. The last arguments may specify distributional parameters for which however default settings exist. If the second argument is specified it must be equal to the first.
'r'	rand(nr,nc,'r'<,rank>)	Creates $nr \times nc$ full rectangular matrix with $rank \leq min(nr, nc)$. The default value for $rank$ is $min(nr, nc)$.
'e'	rand(n,n,'e'<,rank<,elo,ehi>>)	Creates $n \times n$ symmetric matrix with $n - rank$ zero eigenvalues and $nrnk$ nonzero eigenvalues uniformly distributed in the interval $[elo, ehi]$ where at least one of the two interval ranges must be different from zero. The default value for $rank$ is n and the default interval is $[0, 1]$.
'o'	rand(nr,nc,'o')	Creates $nr \times nc$, $nr \geq nc$, column orthogonal rectangular matrix. No further arguments can be used. The entries of the generated orthogonal matrix are in $[-1, 1]$ and their columns sum-of-squares are one.

Matrix Types for `rand` Function

"dist"=	**Add. Arg.**	**Description**
"uni"	a, b	uniform with lower range a and upper range b
"bet"	α, β	Beta, $\mathcal{BE}(\alpha, \beta)$, with $\alpha > 0$ and $\beta > 0$
"bin"	n, p	Binomial, $\mathcal{B}(n, p)$, with $n = 1, 2, \ldots$ and $0 < p < 1$
"cau"	α, β	noncentral Cauchy, $\mathcal{C}(\alpha, \beta)$, with $-\infty < \alpha < \infty$ and $\beta > 0$
"dexp"		Double Exponential, $\mathcal{DE}(\lambda)$,
"exp"	λ	Exponential, $\mathcal{E}(\lambda)$, $\lambda > 0$
"fre"	α	Fréchet, $\mathcal{FR}(\alpha)$, with $\alpha > 0$
"fsn"	α, β	Snedecor's F, $\mathcal{F}(\alpha, \beta)$, with $\alpha > 0$ and $\beta > 0$
"gam"	α, β	Gamma, $\mathcal{G}(\alpha, \beta)$, with $\alpha > 0$ and $\beta > 0$
"geo"	p	Geometric, $\mathcal{GE}(p)$, with $0 < p < 1$
"hyp"	α, β, n	Hypergeometric, $\mathcal{H}(\alpha, \beta, n)$ with $\alpha > 0$, $\beta > 0$, and $1 \leq n \leq \alpha + \beta$
"lon"	μ, σ	Lognormal, $\mathcal{LN}(\mu, \sigma)$, with mean μ and standard deviation $\sigma > 0$
"nbi"	r, p	negative Binomial, $\mathcal{NB}(r, p)$, with $r > 0$ and $0 < p < 1$
"nor"	μ, σ	Normal, $\mathcal{N}(\mu, \sigma)$, with mean μ and standard deviation $\sigma > 0$
"poi"	λ	Poisson, $\mathcal{P}(\lambda)$, with $\lambda > 0$
"ral"	σ	Rayleigh with $\sigma > 0$ (equivalent to Rice when $\mu = 0$)
"ric"	μ, σ	Rice with mean μ and $\sigma > 0$
"tab"	$[p_1, \ldots, p_n]$	tabled probability distribution with given table $0 \leq p_1 \leq p_2 \leq \ldots \leq p_n \leq 1$
"stu"	n	Student's t, $\mathcal{S}(n)$, with $n = 1, 2, \ldots$
"tri"	h	Triangular distribution $0 < h < 1$
"wei"	α, λ	Weibull, $\mathcal{W}(\alpha, \lambda)$, with $\alpha > 0$ and $\lambda > 0$

Distribution Types for `rand` Function

The function `mrand("kind",a,b)` generates vectors or matrices of multivariate random distributions depending on the string argument *kind* and (at most) two more arguments. Currently four choices for *kind* are implemented,

z = mrand("mnor"<,mu,sigma>) Computes n vector `z` of random numbers from the multivariate normal distribution $\mathcal{N}(\mu, \mathbf{\Sigma})$. By default `mu=0` and `sigma=1`.

- When `sigma` is a given lower (or upper) $n \times n$ triangular Cholesky factor $\mathbf{L}$ of a covariance matrix and `mu` is a given n mean vector, the result `z` is computed by:

$$z = \mathbf{L}x + \mu$$

- When `sigma` is a given $n \times n$ (positive definite) symmetric covariance matrix and `mu` is a given n mean vector, the result `z` is computed by:

$$z = \mathbf{L}x + \mu \quad \text{where} \quad \mathbf{\Sigma} = \mathbf{L}\mathbf{L}^T$$

- The `mrand("mnor",mu,sigma)` function was extended to permit the input of an $n \times m$ matrix `mu` with values mu_{ij}. Then an $n \times m$ matrix `z` is returned containing m separate vectors of random numbers from the multivariate normal distributions $\mathcal{N}(\mu_j, \mathbf{\Sigma})$ with different mean vectors but identical covariance matrix. This extension corresponds to m separate calls of the common function may, however, save computer time in some applications.

z = mrand("mnom"<,n,p>) Computes n vector `z` of random numbers from the multinomial distribution $\mathcal{M}(n, p_1, \ldots, p_r)$. There should be $n \geq 2$. By default `n=2` and `p=.5`.

- When `n` is a given scalar, $n \geq 1$, and `p` is an r vector with

$$0 \leq p_i \leq 1 \quad \text{and} \quad \sum_{i=1}^{r} p_i = 1$$

  then the result `z` is an r vector computed using the random generator for the Binomial distribution (see [257], p.224).
- The `mrand("mnom",n,p)` function was extended to permit the input of a m vector `n` with values n_j and a $r \times m$ matrix p_{ij} where value n_j corresponds to column j of p. Then an $r \times m$ matrix `z` is returned containing m separate vectors of random numbers from the multinomial distributions $\mathcal{M}(n_j, p_{1j}, \ldots, p_{rj})$.

z = mrand("unis"<,n,r>) Computes n vector `z` of random numbers uniformly distributed **inside** an n dimensional sphere with radius r. There should be $n \geq 2$. By default `n=2` and `r=1`.

z = mrand("unos"<,n,rad>) Computes n vector `z` of random numbers uniformly distributed **on** an n dimensional sphere with radius r. There should be $n \geq 2$. By default `n=2` and `r=1`.

z = mrand("unie"<,n>) Computes n vector `z` of random numbers uniformly distributed **inside** an n dimensional unit cube

$$(z_1, \ldots, z_n) | \sum_{i=1}^{n} z_i \leq 1, \quad z_i \geq 0,\ i = 1, \ldots, n$$

There should be $n \geq 2$. By default `n=2`.

z = mrand("unoe"<,n>) Computes n vector `z` of random numbers uniformly distributed **on** an n dimensional unit cube

$$(z_1, \ldots, z_n) | \sum_{i=1}^{n} z_i = 1, \quad z_i \geq 0,\ i = 1, \ldots, n$$

There should be $n \geq 2$. By default `n=2`.

Functions shape(a<,nrb><,ncb>), diag(a<,k>), dia2vec(a<,k>), replace(a,old,new<,rel>), and loc(a<,crit<,va

`b = shape(a<,nrb><,ncb>)`

1. This function is useful for creating an $nrb \times ncb$ matrix containing the entries of a given matrix, vector, or scalar object `a`.
2. If both arguments, `a` and `b` are missing, then `b` is set by `a`.
3. If one of the arguments, `nrb` or `ncb` is missing, its default is computed by dividing N by the given other argument, where $N = m * n$ for a given $m \times n$ matrix **A**.
4. Assume, the result `b` has $M = nrb * ncb$ entries. If $M < N$ then only the first M entries of `a` are rowwise moved to `b`. For $M > N$, the entries of `a` are moved in cycles multiple times into `b`.

The following example specifies an input row vector with 18 columns (no separating commas are used) which should be reordered into a 6×3 matrix **B**.

```
a = [ 42.78275      127.62282      1464
      103.26625      20.34239      1591
      -27.39691      90.78732      2553
      -14.31073      31.26545      2540
       -1.16289      84.89600      2158
      -18.13447      30.18118      2352 ];
b = shape(a,.,3);
print "This should be 6 by 3 matrix:", b;
```

The output is as follows:

```
   |         1         2         3
---------------------------------
 1 |     42.78    127.62   1464.00
 2 |    103.27     20.34   1591.00
 3 |    -27.40     90.79   2553.00
 4 |    -14.31     31.27   2540.00
 5 |     -1.16     84.90   2158.00
 6 |    -18.13     30.18   2352.00
```

`b = diag(a<,k>)` This function can be used to create a matrix **B** with one sub- or superdiagonal with entries from a given vector a or a given $m \times n$ matrix **A**. The result depends on the fact whether the specified input argument `a` is a vector or a matrix:

1. for input n vector a:
   (a) For unspecified k or $k = 0$: Returns $n \times n$ diagonal matrix **B** with v in main diagonal.
   (b) For $k > 0$: Returns $(n + k) \times (n + k)$ matrix **B** with v in k-th superdiagonal.
   (c) For $k < 0$: Returns $(n - k) \times (n - k)$ matrix **B** with v in k-th subdiagonal.
2. for $m \times n$ matrix **A**: The $m \times n$ output matrix **B** contains only zero entries except:
   (a) For unspecified k or $k = 0$: **B** contains main diagonal of **A** in main diagonal.
   (b) $k > 0$: **B** contains kth superdiagonal of **A**.
   (c) $k < 0$: **B** contains kth subdiagonal of **A**.

- Input vector v:

```
v = [ 1 2 3 ];
print b = diag(v); /* same as: print b = diag(v,0); */
print b = diag(v,1);
print b = diag(v,-1);
```

```
 D |   1   2   3          U |   1   2   3   4      L |   1   2   3   4
---------------          --------------------     --------------------
 1 |   1                  1 |   0   1   0   0      1 |   0
 2 |       2              2 |       0   2   0      2 |   1   0
 3 |           3          3 |           0   3      3 |   0   2   0
                          4 |               0      4 |   0   0   3   0
```

- Input $m \times n$ matrix **A**:

```
a = [ 1  2  3  4,
      5  6  7  8,
      9 10 11 12];
print b = diag(a); /* same as: print b = diag(a,0); */
print b = diag(a,1); print b = diag(a,-1);
```

```
 D |    1    2    3          D |   1   2          D |    1    2
------------------          -----------          --------------
 1 |    1                    1 |   2              1 |    5
 2 |         6               2 |       7          2 |        10
 3 |             11
```

`b = dia2vec(a<,k>)` This function is useful to move a diagonal, super-, or subdiagonal from an $m \times n$ input matrix $\mathbf{A}$ into a result vector v. The scalar k is optional. The result vector v contains for unspecified k or $k = 0$, the diagonal of $\mathbf{A}$, for $0 < k < n$ the k-th superdiagonal, and for $-m < k < 0$ the $-k$-th subdiagonal of $\mathbf{A}$.

```
a = [ 1  2  3  4,
      5  6  7  8,
      9 10 11 12];
```

1. `v = dia2vec(a)` or `v = dia2vec(a,0)`: returns: `v= [ 1 6 11]`.
2. `u = dia2vec(a,1)` returns: `u= [ 2 7 12]`
3. `w = dia2vec(a,-1)` returns: `w= [ 5 10]`

`b = replace(a,old,new<,rel>)` Sometimes it is necessary to replace all occurences of a value in a matrix or vector by a specific other value. The first argument, *a* spcifies the name of the object, the second argument *old* specifies the object value that has to be changed, and the third argument *new* specifies the new value that replaces the old one. There is an optional fourth argument *rel*, specifying a relationship between the object values and the value *old*. By default this relationship is $rel = 0$, which means equality, i.e. all object entries with value *equal to old* are replaced by the value *new*. The following relationships are possible:

rel	Replace all entries with
0	values == *old*
-1	values <= *old*
1	values >= *old*
-2	values < *old*
2	values > *old*

```
a = [ 1  2  2  3,
      .  4  4  5,
      3  .  .  4 ];

/* replace missing values */
b = replace(a,.,-99);
print b;
```

```
                    |         1         2         3         4
--------------------------------------------------------------
 1 |    1.0000    2.0000    2.0000    3.0000
 2 |  -99.0000    4.0000    4.0000    5.0000
 3 |    3.0000  -99.0000  -99.0000    4.0000
```

```
a = [ 1  2  2  3,
      .  4  4  5,
      3  .  .  4 ];

/* replace all <= 3 with 9 */
c = replace(a,3,9,-1);
print c;
```

```
                    |         1         2         3         4
--------------------------------------------------------------
 1 |   9.00000   9.00000   9.00000   9.00000
 2 |    .        4.00000   4.00000   5.00000
 3 |   9.00000    .         .        4.00000
```

`indx = loc(a<,crit<,val>>)` The function `loc` finds all entries in input object `a` which satisfy a condition specified by the second and third input argument.

The following conditions may be specified:

Criterion	Meaning
`"nonz"`	find locations of all nonzeros (includes locations of missing value) this is the default and must not be specified
`"zero"`	find locations of all zeros
`"miss"`	find locations of all missing values
`"nznm"`	find locations of all nonzeros and nonmissings
`"great"`	find locations of all entries in `a` which are larger than a specified third argument
`"small"`	find locations of all entries in `a` which are smaller than a specified third argument
`"equal"`	find locations of all entries in `a` which are equal to a specified third argument
`"unequ"`	find locations of all entries in `a` which are unequal to a specified third argument

The second and third argument must not be specified for the `"nonz"` condition. The third argument must not be specified for the `"nonz"`, `"zero"`, `"miss"`, `"nznm"` conditions. If the third argument is not be specified for the `"great"`, `"small"`, `"equal"`, `"nonequ"` conditions, a numeric value of zero is used as the default. If the third argument is specified as a string, the `"great"` and `"small"` conditions concern only the length of the string.

The function `loc` returns with `indx` either

1. a misssing value if there are no entries in the object `a` which satify the specified condition or
2. a scalar index if there is only one entry in `a` which satisfies the condition or
3. a vector of K indices, when $K > 1$ entries of `a` satisfy the condition and the input object `a` is a vector, or
4. a $K \times 2$ matrix of row and column indices, when $K > 1$ entries of `a` satisfy the condition and the input object `a` is a matrix.

1. Some simple examples:

```
w = [ -1. 0. 0. 1. 2. 3. . ];
ind1 = loc(w);
print "ind1=", ind1;
```

```
ind1=
   |   1
--------
 1 |   1
 2 |   4
 3 |   5
 4 |   6
 5 |   7
```

```
ind2 = loc(w,"nznm");
print "ind2=", ind2;
```

```
ind2=
   |   1
--------
 1 |   1
 2 |   4
 3 |   5
 4 |   6
```

```
ind3 = loc(w,"zero");
print "ind3=", ind3;
```

```
ind3=
   |   1
--------
 1 |   2
 2 |   3
```

```
  ind4 = loc(w,"miss",2.);
  print "ind4=", ind4;
```

```
ind4=
   |   1
--------
 1 |   7
```

```
  ind5 = loc(w,"great",2.);
  print "ind5=", ind5;
```

```
ind5=
   |   1
--------
 1 |   6
```

```
str = [ "here" "there" "here" "there" ];
ind = loc(str,"equal","here");
print "ind=", ind;
```

```
ind=
   |   1
--------
 1 |   1
 2 |   3
```

```
  a = [ 2.  3.  4.  5.  6.,
        4.  4.  5.  6.  7.,
        0.  3.  6.  7.  8.,
        0.  0.  2.  8.  9.,
        0.  0.  0.  1. 10. ];
  ind = loc(a);
  print "Ind=", ind;
```

```
Ind=
   |   1   2
------------
 1 |   1   1
 2 |   1   2
 3 |   1   3
 4 |   1   4
 5 |   1   5
 6 |   2   1
 7 |   2   2
 8 |   2   3
 9 |   2   4
10 |   2   5
11 |   3   2
12 |   3   3
13 |   3   4
14 |   3   5
15 |   4   3
16 |   4   4
17 |   4   5
18 |   5   4
19 |   5   5
```

5.3.4 Matrix Definition by Concatenation

There are six different types of matrix concatenations available in CMAT:

Op	Direction		a Op b
`->`	horizontally	left-to-right	$[a \quad b]$
`<-`	horizontally	right-to-left	$[b \quad a]$
`|>`	vertically	top-to-bottom	$\begin{bmatrix} a \\ b \end{bmatrix}$
`<|`	vertically	bottom-to-top	$\begin{bmatrix} b \\ a \end{bmatrix}$
`\>`	diagonally	left-top-to-right-bottom	$\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$
`</`	diagonally	left-bottom-to-right-top	$\begin{bmatrix} 0 & b \\ a & 0 \end{bmatrix}$

Matrix Concatenation Operators

The following statements generate row vectors `a1` and `a2`, column vectors `a3` and `a4`, and diagonal matrices `a5` and `a6`:

```
a1 = a2 = a3 = a4 = a5 = a6 = 0;
for (i = 1; i <= 3; i++) a1 = a1 -> i;
for (i = 1; i <= 3; i++) a2 = a2 <- i;
for (i = 1; i <= 3; i++) a3 = a3 |> i;
for (i = 1; i <= 3; i++) a4 = a4 <| i;
for (i = 1; i <= 3; i++) a5 = a5 \> i;
for (i = 1; i <= 3; i++) a6 = a6 <\ i;
print a1, a2, a3, a4, a5, a6;
```

The two row vectors `a1` and `a2` and the two column vectors `a3` and `a4` are

```
   |   1   2   3   4
-------------------
 1 |   0   1   2   3

   |   1   2   3   4
-------------------
 1 |   3   2   1   0
```

```
   |   1
--------
 1 |   0
 2 |   1
 3 |   2
 4 |   3
```

```
   |   1
--------
 1 |   3
 2 |   2
 3 |   1
 4 |   0
```

and CMAT finds that `a5` is diagonal and `a6` is sparse matrix, that means, CMAT needs only $O(n)$ memory for each.

```
 D |   1   2   3   4
-------------------
 1 |   0
 2 |       1
 3 |           2
 4 |               3
```

```
   |   1   2   3   4
-------------------
 1 |   0   0   0   3
 2 |   0   0   2   0
 3 |   0   1   0   0
 4 |   0   0   0   0
```

Similarily, we are also able to concatenate string data.

```
ss = s1 = "string";
for (i = 0; i < 3; i++) ss = ss \> s1;
print ss;
```

```
 D |        1        2        3
-------------------------------
 1 |   string
 2 |            string
 3 |                     string
```

5.3.5 Two Types of Index Processing in Matrices

The following situations may occur for accessing entries of a matrix `a[m,n]`:

1. there are two scalar indices, e.g. i and j: `a[i,j]` refers to only one entry in row i and column j of matrix `a`
2. there is one scalar i and one k vector v index: e.g. `a[i,v]` refers to k entries in row i and the k columns specified by the entries of v
3. there are two vector indices, e.g. u and v, where u has k entries and v has l entries: `a[u,v]` then refers to $k * l$ entries in in the rows specified by entries of u and the columns specified by the entries of v
4. there is only one index: this must be either
   - one 2-vector index v: `a[v]` refers to one entry in the row specified by $v[1]$ and the column specified by $v[2]$ or
   - one 2-column matrix m with k rows: `a[m]` refers to k entries in the rows specified by the first column of m and the columns specified by the second column of m

Here are a few examples:

1. Use of `loc` function to obtain 2-column index matrix:

```
a = [ 2.  3.  4.  5.  6.,
      4.  4.  5.  6.  7.,
      0.  3.  6.  7.  8.,
      0.  0.  2.  8.  9.,
      0.  0.  0.  1. 10. ];
ind = loc(a);
print "Ind=", ind;
```

```
Ind=
    |   1   2
------------
  1 |   1   1
  2 |   1   2
  3 |   1   3
  4 |   1   4
  5 |   1   5
  6 |   2   1
  7 |   2   2
  8 |   2   3
  9 |   2   4
 10 |   2   5
 11 |   3   2
 12 |   3   3
 13 |   3   4
 14 |   3   5
 15 |   4   3
 16 |   4   4
 17 |   4   5
 18 |   5   4
 19 |   5   5
```

Note, that `c` is a vector and `d` is the same as `a`:

```
c = a[ind];
print "C=", c;
d[ind] = c;
print "d=", d;
```

```
C=
   |         1
---------------
 1 |    2.0000
 2 |    3.0000
 3 |    4.0000
 4 |    5.0000
 5 |    6.0000
 6 |    4.0000
 7 |    4.0000
 8 |    5.0000
 9 |    6.0000
10 |    7.0000
11 |    3.0000
12 |    6.0000
13 |    7.0000
14 |    8.0000
15 |    2.0000
16 |    8.0000
17 |    9.0000
18 |   1.00000
19 |   10.0000
```

```
d=
   |         1         2         3         4         5
-----------------------------------------------------
 1 |    2.0000    3.0000    4.0000    5.0000    6.0000
 2 |    4.0000    4.0000    5.0000    6.0000    7.0000
 3 |   0.00000    3.0000    6.0000    7.0000    8.0000
 4 |   0.00000   0.00000    2.0000    8.0000    9.0000
 5 |   0.00000   0.00000   0.00000   1.00000   10.0000
```

2. Difference between resetting obejcts and reusing them:

   This copies objects after freeing old content:

```
a = [ 1 2 3 ,
      2 3 1 ,
      3 1 2 ];
b = [ 1 2 3 ,
      0 2 3 ,
      0 0 3 ];
c = [ 1 0 0 ,
      2 3 0 ,
      1 2 3 ];


x = y = z = cons(5,5);
x = a;
y = b;
z = c;
print "x,y,z=", x,y,z;
```

```
x,y,z=
 S |   1   2   3
---------------
 1 |   1
 2 |   2   3
 3 |   3   1   2

 U |   1   2   3
---------------
 1 |   1   2   3
 2 |   0   2   3
 3 |   0   0   3

 L |   1   2   3
---------------
 1 |   1   0   0
 2 |   2   3   0
 3 |   1   2   3
```

   This copies objects inside old content:

```
x = y = z = cons(5,5);
x[1,1] = a;
y[1,1] = b;
z[1,1] = c;
print "x,y,z=", x,y,z;
```

```
x,y,z=

 S |   1   2   3   4   5
-----------------------
 1 |   1
 2 |   2   3
 3 |   3   1   2
 4 |   0   0   0   0
 5 |   0   0   0   0   0


 U |   1   2   3   4   5
-----------------------
 1 |   1   2   3   0   0
 2 |   0   2   3   0   0
 3 |   0   0   3   0   0
 4 |   0   0   0   0   0
 5 |   0   0   0   0   0

 L |   1   2   3   4   5
-----------------------
 1 |   1   0   0   0   0
 2 |   2   3   0   0   0
 3 |   1   2   3   0   0
 4 |   0   0   0   0   0
 5 |   0   0   0   0   0
```

Special properties are no longer valid:

```
x = y = z = cons(5,5);
x[2,1] = a;
y[2,1] = b;
z[2,1] = c;
print "x,y,z=", x,y,z;
```

```
x,y,z=
   |   1   2   3   4   5
-----------------------
 1 |   0   0   0   0   0
 2 |   1   2   3   0   0
 3 |   2   3   1   0   0
 4 |   3   1   2   0   0
 5 |   0   0   0   0   0


   |   1   2   3   4   5
-----------------------
 1 |   0   0   0   0   0
 2 |   1   2   3   0   0
 3 |   0   2   3   0   0
 4 |   0   0   3   0   0
 5 |   0   0   0   0   0

 L |   1   2   3   4   5
-----------------------
 1 |   0   0   0   0   0
 2 |   1   0   0   0   0
 3 |   2   3   0   0   0
 4 |   1   2   3   0   0
 5 |   0   0   0   0   0
```

3. Use of 2-column index matrix on left side:

   Loading with 2 column index matrix: (Note, matrix is extended to 6 columns)

```
a = [ 1 2 3 ,
      2 3 1 ,
      3 1 2 ];
b = [ 1 2 3 ,
      0 2 3 ,
      0 0 3 ];
c = [ 1 0 0 ,
      2 3 0 ,
      1 2 3 ];
d = [ 3. 5. 7. ];
x = y = z = cons(5,5);
ind = [ 1 2 ,
        3 4 ,
        5 6 ];
x[ind] = a;
y[ind] = b;
z[ind] = c;
print "x,y,z=", x,y,z;
```

```
x,y,z=
 U |   1   2   3   4   5   6
---------------------------
 1 |   0   1   0   0   0   0
 2 |   0   0   0   0   0   0
 3 |   0   0   0   2   0   0
 4 |   0   0   0   0   0   0
 5 |   0   0   0   0   0   3

 U |   1   2   3   4   5   6
---------------------------
 1 |   0   1   0   0   0   0
 2 |   0   0   0   0   0   0
 3 |   0   0   0   2   0   0
 4 |   0   0   0   0   0   0
 5 |   0   0   0   0   0   3

 U |   1   2   3   4   5   6
---------------------------
 1 |   0   1   0   0   0   0
 2 |   0   0   0   0   0   0
 3 |   0   0   0   0   0   0
 4 |   0   0   0   0   0   0
 5 |   0   0   0   0   0   0
```

```
print "Duplicate setting at same location";
d = [ 3. 5. 7. ];
x = cons(5,5);
ind = [ 1 2 ,
        3 4 ,
        3 4 ];
x[ind] = d;
print "x=", x;
```

```
Duplicate setting at same location

x=
 U |         1         2         3         4         5
-----------------------------------------------------
 1 |   0.00000    3.0000   0.00000   0.00000   0.00000
 2 |         0   0.00000   0.00000   0.00000   0.00000
 3 |         0         0   0.00000    7.0000   0.00000
 4 |         0         0         0   0.00000   0.00000
 5 |         0         0         0         0   0.00000
```

5.3.6 Matrix Definition by Indexed Assignments

By default, CMAT works with 1 based indices since most people are more used to 1 based indices than to the C style 0 based indices. However, since CMAT should be mostly compatible to C, at least in the scalar case, the option `INDBASE=0` can be used to run CMAT with zero based indices. Although it is always possible during execution to switch between zero and one based indices, however, the user should be aware that this could be a source of trouble.

Indexed Assignments `B[irb,icb] = A[ira,ica];`

The most general case of an indexed assignment is

```
B[irb,icb] = A[ira,ica];
```

where `ira, ica, irb`, and `icb` can be defined as

1. nonnegative integer scalar (index): The indices `ira` and `ica` must be inside the valid dimensions of matrix `A` (depending on the specified index base). The indices `irb` and `icb` may be larger than the prior row resp. column dimensions of `B`.
2. index vector literal: That means inside brackets `[` and `]` a list of indices containing colon and pound operators and identifiers may be used.
3. identifier: That means it can be the name of a vector containing valid index entries.
4. missing value: There are two valid forms, either a period is used or the index is simply skipped.

The execution of the statement `B[irb, icb] = A[ira, ica];` is identical to the execution of the two statements:

```
C = A[ira,ica]; B[irb,icb] = C;
```

Pulling down the object `C=A[ira,ica];` seems to be straight forward. The object **A** must be prior defined and the index sets `ira` and `ica` must be in the scope of the defined **A**. If the vector or scalar `ira` has n_r entries and the vector or scalar `ira` has n_c entries then **C** is $n_r \times n_c$ matrix. But, there are different forms of **B** resulting from `B[irb,icb] = C;` and depending on the prior content of `C`, `B`, `irb`, and `irc`.

Indexed Assignment of Form: `B[irb,icb] = C;`

Assignment of Scalar Value `C`: When the right hand side `C` is a scalar, then `C` is moved into all specified locations. If both `irb` and `icb` are index vectors, it is moved to the product locations `irb` × `icb`. If one or both of `irb` and `icb` are missing, it is replaced by all indices of the corresponding dimension of `B`.

Attachment for Scalar Index: For example, if the two indices `irb` and `icb` are scalar, all rows or columns of **C** are attached at (pinned on) the location `B[irb,icb]`. The attachment may extend the old dimensions of `B`. The new dimension of `B` is the maximum of the old dimensions and the dimension of **C** added to the scalar index.

Selection for Index Vectors `irb` and `icb`:

Assigment of Vector `C`: The two index vectors should have the same length as the vector `C` and their pairs specify the index locations where the entries of `C` are being moved to.

Assigment of Matrix `C`: The row dimension of `C` should be equal the number of indices in `irb` and the number of columns should be equal the number of indices in `icb`. The entries of `C` are moved to the index product locations `irb` × `icb`.

Those rows (or columns) of `B` which are not targeted by the index vector remain unchanged. The new dimension of `B` is defined by the maximum of the old dimension of `B` and the maximum index specified in the index vector.

Stacking for missing value: When using @ for one or both of the indices, the object `A` is stacked multiple times into this dimension of `B` and the new dimension of `B` is the product between the old dimension of `B` and the dimension of `C`.

Warning: The result of a statement of form `B[irb,icb] = C;` depends very much on the form of `B` before this statement is executed. Note, that you can use the `free` statement to free `B` before executing the statement to prevent the stacking action.

The following code creates a column vector printed here as row vector:

```
for (i = 1; i <= 9; i++) ind1[i] = i;
print ind1‘;
```

```
   |   1   2   3   4   5   6   7   8   9
---------------------------------------
 1 |   1   2   3   4   5   6   7   8   9
```

This approach of generating a vector is computationally costly, since for each new assignment memory must be released and new allocated. For larger vectors, when efficiency is an issue, it pays off first to define the vector by allocating the memory using the `cons` function before performing the `for` loop,

```
ind2 = cons(9,.,1);
for (i = 1; i <= 9; i++) ind2[i] = i;
print ind2‘;
```

Note that the initialization `ind2 = cons(9);` would not be sufficient for a more efficient specification, since CMAT decides for sparse storage whenever possible. The generated vector with nine zero elements therefore would not need any data memory and the cycle of freeing and releasing memory while performing the `for` statement would still be performed. The vector initialization with a nonzero element prevents the memory management in the succeeding `for` loop.

However, another even more efficient way to obtain a row vector with the same entries, uses the colon notation:

```
ind3 = [1 : 9];
print ind3;
```

An important application is the inverse permutation of a given order:

```
c = [1 : 5];                              |   1   2   3   4   5
ind = [3 2 4 5 1];                       ----------------------
print b[ind] = c;                         1 |   5   2   1   3   4
```

Now, we move a numeric matrix into a matrix of string data by applying the *selection* property of index vectors,

```
a66 = [ "a11" : "a16",          |     1     2     3     4     5     6
        "a21" : "a26",         ---------------------------------------
        "a31" : "a36",          1 |     9   a12   a13     7   a15     8
        "a41" : "a46",          2 |   a21   a22   a23   a24   a25   a26
        "a51" : "a56",          3 |   a31   a32   a33   a34   a35   a36
        "a61" : "a66" ];        4 |     3   a42   a43     1   a45     2
f = a66;                        5 |   a51   a52   a53   a54   a55   a56
iv3 = [ 4 6 1 ];                6 |     6   a62   a63     4   a65     5
f[iv3,iv3] = [ 1 2 3,
               4 5 6,
               7 8 9 ];
print f;
```

When using scalar indices we *attach* a new matrix at a specified location,

```
b = cons(4,4,1);                |     1     2     3     4     5
a23 = [ "a11" : "a13",         ----------------------------------
        "a21" : "a23" ];        1 |     1     1     1     1     0
b[2,3]= a23;                    2 |     1     1   a11   a12   a13
print b;                        3 |     1     1   a21   a22   a23
                                4 |     1     1     1     1     0
```

The next example illustrates the *stacking* property when using missing value indices.

```
a2 = [ "a1" "a2" ];             |    1    2    3    4
f = cons(2,2,.);               ------------------------
f[@,@] = a2;                    1 |   a1   a2   a1   a2
print f;                        2 |   a1   a2   a1   a2
f = cons(2,2,.);
f[.,.] = a2;                    |    1    2
print f;                       --------------
f = a2;                         1 |   a1   a2
print f;                        2 |   a1   a2

                                |    1    2
                               --------------
                                1 |   a1   a2
```

You may also use the three-element colon construction, for example, with floating point values.

```
r = [ 20.5 : -5 : -10 ];
print r;
```

```
   |         1         2         3         4
---------------------------------------------
 1 |   20.5000   15.5000   10.5000    5.5000

   |         5         6         7
-----------------------------------
 1 |    0.5000   -4.5000   -9.5000
```

You can also extend a scalar, vector, or matrix which is already defined to a (larger) vector or matrix. The following is a scalar to vector extension:

```
d = 1111;
for (i = 2; i <= 8; i++) d[i] = i;
print d‘;
```

```
   |      1      2      3      4      5      6      7      8
-------------------------------------------------------------
 1 |   1111      2      3      4      5      6      7      8
```

Assignment statements with indexed variables on the left-hand side, for example $\mathbf{B}_{ij}$, and matrices on the right-hand side are executed so that the right-hand side matrix is attached with its first element at the location `[i,j]` of matrix **B**. The attachment may exceed the row and column numbers of the former definition of **B**. The extended **B** matrix is filled up with zeros.

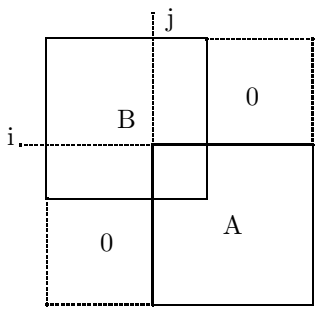

Fig. 1 Attach Matrix A at $\mathbf{B}_{ij}$

For example, the following input results in the 4×3 matrix **B**

```
b = [ 1 2,
      3 4,
      5 6];
a = [ 9 8,
      7 9];
b[3,2] = a;
print b;
```

```
   |   1   2   3
---------------
 1 |   1   2   0
 2 |   3   4   0
 3 |   5   9   8
 4 |   0   7   9
```

The location `[i,j]` of attachment may be even outside the current definition of the left-hand side matrix

```
b = [ 1 2,
      3 4];
a = [ 9 8,
      7 9];
b[4,3] = a;
print b;
```

```
   |   1   2   3   4
-------------------
 1 |   1   2   0   0
 2 |   3   4   0   0
 3 |   0   0   0   0
 4 |   0   0   9   8
 5 |   0   0   7   9
```

Matrix Assignments `B[irb,icb] = A[imat];` **or** `B[jmat] = A[ira,ica];` **or** `B[jmat] = A[imat];`

When using an $m \times 2$ index matrix `imat` or `jmat` instead of two index vectors separated by comma we access only a specific set of m index pairs (i, j) for matrix **A** or **B** instead of the Kronecker product of the two index sets. See page 48 for more detail.

5.4 Defining Tensors

5.4.1 Tensor Definition by Type Declaration

The type declarations `int`, `real`, `complex` and `char` can be used to allocate memory for tensors with constant values. If the declaration does not contain an explicit initialization by a constant or a scalar identifier, the entries are initialized to zero.

```
print "Tensor definition by type declaration";
int A2[2,3,4]=9;
print "A2[2,3,4]=", A2;
```

Tensors are printed as lists of matrices. Here there is tensor `a2` equivalent to a list two 3×4 matrices:

```
**************************
Tensor A2 with 3 Dimensions
**************************


            A2_1
            ****

   Dense Matrix (3 by 4)

  |       1      2      3      4
--------------------------------
1 |       9      9      9      9
2 |       9      9      9      9
3 |       9      9      9      9


            A2_2
            ****

   Dense Matrix (3 by 4)

  |       1      2      3      4
--------------------------------
1 |       9      9      9      9
2 |       9      9      9      9
3 |       9      9      9      9
```

```
real B2[3,4,5] = 99.99;
print "B2[3,4,5]=", B2;
```

```
**************************
Tensor B2 with 3 Dimensions
**************************


                        B2_1
                        ****

                   Dense Matrix (4 by 5)

  |           1          2          3          4          5
--------------------------------------------------------------
1 |   99.990000  99.990000  99.990000  99.990000  99.990000
2 |   99.990000  99.990000  99.990000  99.990000  99.990000
3 |   99.990000  99.990000  99.990000  99.990000  99.990000
4 |   99.990000  99.990000  99.990000  99.990000  99.990000


                        B2_2
                        ****

                   Dense Matrix (4 by 5)

  |           1          2          3          4          5
--------------------------------------------------------------
1 |   99.990000  99.990000  99.990000  99.990000  99.990000
2 |   99.990000  99.990000  99.990000  99.990000  99.990000
3 |   99.990000  99.990000  99.990000  99.990000  99.990000
4 |   99.990000  99.990000  99.990000  99.990000  99.990000


                        B2_3
                        ****

                   Dense Matrix (4 by 5)

  |           1          2          3          4          5
--------------------------------------------------------------
1 |   99.990000  99.990000  99.990000  99.990000  99.990000
2 |   99.990000  99.990000  99.990000  99.990000  99.990000
3 |   99.990000  99.990000  99.990000  99.990000  99.990000
4 |   99.990000  99.990000  99.990000  99.990000  99.990000
```

The `dim` function is a straight forward extension of the `nrow` and `ncol` functions and can be used for obtaining the sizes of dimensions of vectors, matrices, tensors, and lists:

```
n2 = dim(B2);
print "Dimension B2=", n2;
```

```
Dimension B2=
   |   1   2   3
---------------
 1 |   3   4   5
```

```
  n3 = dim(B2,3);
  print "Number columns of B2=", n3;
```

```
Number columns of B2= 5
```

5.4.2 Tensor Definition by Functions

The following functions can be used for defining tensors: `const(`$nind <, val >$`)` and `randt(`$nr, nc, type, \ldots$`)`.

Function `t = const(`$nind <, val >$`)`

The function `t = const(`$nind <, val >$`)` is a tensor extension of the matrix function `m = cons(`$nr <, nc, val, type >$`)`. Assuming the tensor should have K dimensions the input vector `nind` should specify K integers as the upper index ranges of each dimension. The upper ranges of the K dimensions should only be the assumed lower bounds, using statements the ranges can still be increased later, by just defining tensor entries outside the specified ranges.

```
  nind  = cons(3,1,4);    /* cons(nr,nc,val) */
  atens = const(nind,.9);
  print "Constant Tensor=", atens;
```

Function `t = randt(`$nind <, type, \ldots >$`)`

The function `t = randt(`$nind <, type, \ldots >$`)` is a tensor extension of the matrix function `m = rand(`$< nr, nc, type, \ldots >$`)`. Assuming the tensor should have K dimensions the input vector `nind` should specify K integers as the upper index ranges of each dimension. The upper ranges of the K dimensions should only be the assumed lower bounds, using statements the ranges can still be increased later, by just defining tensor entries outside the specified ranges.

```
  nind  = cons(3,1,4);    /* cons(nr,nc,val) */
  btens = randt(nind,"duni");
  print "Random Tensor=", btens;
```

The distribution functions are the same as for the `rand()` function. See page 40 for a table of distribution functions:

"dist"=	**Add. Arg.**	**Description**
"uni"	a, b	uniform with lower range a and upper range b
"bet"	α, β	Beta, $\mathcal{BE}(\alpha, \beta)$, with $\alpha > 0$ and $\beta > 0$
"bin"	n, p	Binomial, $\mathcal{B}(n, p)$, with $n = 1, 2, \ldots$ and $0 < p < 1$
"cau"	α, β	noncentral Cauchy, $\mathcal{C}(\alpha, \beta)$, with $-\infty < \alpha < \infty$ and $\beta > 0$
"dexp"		Double Exponential, $\mathcal{DE}(\lambda)$,
"exp"	λ	Exponential, $\mathcal{E}(\lambda)$, $\lambda > 0$
"fre"	α	Fréchet, $\mathcal{FR}(\alpha)$, with $\alpha > 0$
"fsn"	α, β	Snedecor's F, $\mathcal{F}(\alpha, \beta)$, with $\alpha > 0$ and $\beta > 0$
"gam"	α, β	Gamma, $\mathcal{G}(\alpha, \beta)$, with $\alpha > 0$ and $\beta > 0$
"geo"	p	Geometric, $\mathcal{GE}(p)$, with $0 < p < 1$
"hyp"	α, β, n	Hypergeometric, $\mathcal{H}(\alpha, \beta, n)$ with $\alpha > 0$, $\beta > 0$, and $1 \leq n \leq \alpha + \beta$
"lon"	μ, σ	Lognormal, $\mathcal{LN}(\mu, \sigma)$, with mean μ and standard deviation $\sigma > 0$
"nbi"	r, p	negative Binomial, $\mathcal{NB}(r, p)$, with $r > 0$ and $0 < p < 1$
"nor"	μ, σ	Normal, $\mathcal{N}(\mu, \sigma)$, with mean μ and standard deviation $\sigma > 0$
"poi"	λ	Poisson, $\mathcal{P}(\lambda)$, with $\lambda > 0$
"ral"	σ	Rayleigh with $\sigma > 0$ (equivalent to Rice when $\mu = 0$)
"ric"	μ, σ	Rice with mean μ and $\sigma > 0$
"tab"	$[p_1, \ldots, p_n]$	tabled probability distribution with given table $0 \leq p_1 \leq p_2 \leq \ldots \leq p_n \leq 1$
"stu"	n	Student's t, $\mathcal{S}(n)$, with $n = 1, 2, \ldots$
"tri"	h	Triangular distribution $0 < h < 1$
"wei"	α, λ	Weibull, $\mathcal{W}(\alpha, \lambda)$, with $\alpha > 0$ and $\lambda > 0$

Distribution Types for `randt` Function

5.4.3 Tensor Definition by Indexed Assignments

All of the remarks for defining matrices by indexed assignments are straightforward generalized to tensors, just by adding more indexed dimensions. This includes the INDBASE option.

Indexed Assignments `B[`$i1, \ldots$`] = A[`$j1, \ldots$`];`

The most general case of an indexed assignment is

`B[`$i1, \ldots$`] = A[`$j1, \ldots$`];`

where $i1, \ldots$, and $j1, \ldots$ can be defined as

1. nonnegative integer scalar (index): The indices $j1, \ldots$ must be inside the valid dimensions of a data object `A` (depending on the specified index base). The indices $i1, \ldots$ may be larger than the prior definition of `B`. The dimensionalities of **A** and **B** must not be the same, however, the dimensionality of **A** should not exceed the dimensionality of **B**.
2. index vector literal: That means inside brackets `[` and `]` a list of indices containing colon and pound operators and identifiers may be used.
3. identifier: That means it can be the name of a vector containing valid index entries.

4. missing value: There are two valid forms, either a period is used or the index is simply skipped.

The execution of the statement `B[`$i1,\ldots$`] = A[`$j1,\ldots$`];` is identical to the execution of the two statements:

`C = A[`$j1,\ldots$`]; B[`$i1,\ldots$`] = C;`

Pulling down the object `C=A[`$j1,\ldots$`]` seems to be straight forward. The object **A** must be prior defined and the index sets $j1, j2, \ldots$ must be in the scope of the defined **A**.

If the vector or scalar j_1 has n_1 entries, the vector or scalar j_2 has n_2 entries etc. then **C** is $n_1 \times n_2 \times \ldots$ tensor. But, there are different forms of **B** resulting from `B[`$i1,\ldots$`] = C;` and depending on the prior content of `C`, `B`, and the index sets $i1, \ldots$.

Indexed Assignment of Form: `B[`$i1,\ldots$`] = C;`

Assignment of Scalar Value `C`: When the right hand side of `C` is a scalar, then `C` is moved into all specified locations. If some of the indices $i1, \ldots$ are index vectors, `C` is moved to the product locations. Missing indices $i1, \ldots$ may occur if **B** is prior defined. Missing indices are then replaced by all indices of the corresponding dimension of the prior defined `B`.

Attachment for Scalar Index: For example, if all the indices $i1, \ldots$ are scalar, the entire data object **C** is attached at (pinned on) the location `B[`$i1,\ldots$`]`. The attachment may extend the old dimensions of `B`. The new dimension of `B` is the maximum of the old dimensions and the dimension of **C** added to the scalar index.

Index Vectors in $i1, \ldots$ and **C** is Object: Let n_i be the number of entries defined by the product of indices $i1 \times i2 \times \ldots$ and n_c be the number of entries of the data object (vector, matrix, tensor) **C**. For $n_i = n_c$ all entries of `C` are moved to **B** using the first index in the outermost and the last index in the innermost loop (for matrices rowwise). For $n_i > n_c$, only the first part of the entries of **C** is moved to the index locations in **B**. For $n_i < n_c$ all entries of **C** are moved cyclic into the index locations of **B**.

Stacking for missing value: When using @ for one or both of the indices, the object `A` is stacked multiple times into this dimension of `B` and the new dimension of `B` is the product between the old dimension of `B` and the dimension of `C`.

Tensor Assignments `B[`$i1,\ldots$`] = A[imat]; or B[jmat] = A[`$j1,\ldots$`]; or B[jmat] = A[imat];`

Assuming **A** or **B** are tensors with dimensionality K. When using an $m \times K$ index matrix `imat` or `jmat` instead of K index vectors separated by comma we access only a specific set of m index pairs $(i1, \ldots)$ for that tensor instead of the Kronecker product of the K index sets. This is a straight forward extension of the similar matrix assignments, see page 48 for more detail.

5.5 Defining Lists

5.5.1 Introduction: Tensors and Lists

The `list name[dim];` statement is used for creating a data list with `dim` entries which are iniutialized as scalars set with missing values.

Data lists may contain the following data objects:

- numerical (int, real, complex) and string scalars
- vectors and matrices
- tensors
- structs
- (sub-)lists (this feature practically extends the one dimensional data lists to two dimensional lists)

Three dimensional tensors are lists of matrices with all equal row and column number.The major difference. You may use functions `mat2ten()` and `ten2mat()` for transforming lists of matrices into tensors and vice versa.

The difference between data lists and data structs is that list entries are referred to with indices (inside brackets) whereas struct entries are referred to with names separated from the structure name with a dot.

Since lists are (currently) restricted to onedimensional arrays of data objects, the extension that lists may contain sublists, is practically equivalent to mutidimensional lists. A nice example is the `parm` return argument of the `cfa()` function, which may be:

1. if no standard errors are computed and for only one sample: this is a list of three objects (F, P, U)
2. if no standard errors are computed and for $k > 1$ samples: these are s lists each with three objects (F, P, U)
3. if standard errors are computed and for only one sample: these are 5 (or 9 for additional robust estimation) lists (estimates, ASEs, t values, lower and upper CIs) each with three objects (F, P, U)
4. if standard errors are computed and for s samples: these are 5 (or 9) lists (estimates, ASEs, t values, lower and upper CIs) of s lists each with three objects (F, P, U), in total this are $5 * s * 3$ (or $9 * s * 3$) lists in three dimensions $[5, s, 3]$ (or $[9, s, 3]$).

The following example illustrates how to create a list of variable names, how to attach it to the dimensions of a tensor, and how to move it from a tensor back into a list of string vectors.

```
srand(1);
nind  = [ 2 3 4 ];
atens = randt(nind,"duni");

list tnam[3];
tnam[1] = [ "time1" "time2" ];
tnam[2] = [ "row1" "row2" "row3" ];
tnam[3] = [ "col1" "col2" "col3" "col4" ];
print "List tnam=",tnam;
```

```
List tnam=
                      ************************
                      List tnam with 3 Entries
                      ************************


                              tnam[1]
                              *******

                         Dense Row Vector (ncol=2)

                            R |        1       2
                                    time1   time2


                              tnam[2]
                              *******

                         Dense Row Vector (ncol=3)

                          R |       1      2      3
                                  row1   row2   row3


                              tnam[3]
                              *******

                         Dense Row Vector (ncol=4)

                       R |       1      2      3      4
                               col1   col2   col3   col4


  ctens = dimname(atens,tnam);
  print "Ctens=",ctens;


Ctens=
                     ******************************
                     Tensor ctens with 3 Dimensions
                     ******************************


                              ctens_1
                              *******

                           Dense Matrix (3 by 4)

                  |        col1       col2       col3       col4
```

```
          ------------------------------------------------
        row1 |   0.1849626  0.9700887  0.3998243  0.2593986
        row2 |   0.9216026  0.9692773  0.5429792  0.5316917
        row3 |   0.0497940  0.0665666  0.8193186  0.5238705


                            ctens_2
                            *******

                       Dense Matrix (3 by 4)

             |        col1       col2       col3       col4
          ------------------------------------------------
        row1 |   0.8533943  0.0671846  0.9570239  0.2971940
        row2 |   0.2726118  0.6899296  0.9767649  0.2265075
        row3 |   0.6882366  0.4127639  0.5585541  0.2872256


  /* get the list back: should be the same as tnam */
  vnam = dimname(ctens);
  print "Original names=",vnam;


Original names=
                         ************************
                         List vnam with 3 Entries
                         ************************


                                 vnam[1]
                                 *******

                        Dense Column Vector (nrow=2)

                            C |        1       2
                                   time1   time2


                                 vnam[2]
                                 *******

                        Dense Column Vector (nrow=3)

                          C |       1      2      3
                                 row1   row2   row3


                                 vnam[3]
                                 *******
```

```
Dense Column Vector (nrow=4)

C |       1      2      3      4
       col1   col2   col3   col4
```

The technical report: *Tensor and List Operations in CMAT* illustrates connections between list and tensor operations.

5.5.2 Syntax for Lists

Before using a list the name must be defined as a list name by the keyword statements:

$$\texttt{list}\ list\_name[irange] <, list\_name2[irange2], \ldots >;$$

whereas the *list_name* specifies the name of the list and the *irange* must be an integer defining the upper index range, i.e. the number of list entries. Note, when the *list_name* was used in CMAT code before this definition, the former object is free'd by default and its content is lost, even if the name was already defined as a list.

After a name was defined as a list, the list entries can be defined in the usual vector notation:

$$list\_name[index] = name\_of\_data\_object$$

.

List entries can be either scalars, vectors, matrices, tensors, structs, or lists (and so permitting the functionality of more than one-dimensional lists). Note, currently only one index can be used, i.e. lists are vectors of objects. Undefined list entries can be referred to as missing values. List entries can be used in arithmetic operations, as function arguments, and in `return();` statements of user functions.

5.5.3 Lists and User Specified Functions

List names can be arguments of user specified functions:

```
print " User Function with list argument";  This should be 15:  15
 function tff1(lst1) {
    a = lst1[1]; b = lst1[2];
    k = (a < b) ? -a : -b;
    a = a + b;
    return(k);
 }

 list lst1;
 lst1[1] = -15; lst1[2] = 27;
 c = tff1(lst1);
 print " This should be 15: ",c;
```

List names can be returned by user specified functions:

```
print "User Function returns with List";     This should be 15:  15
 function tff3(a,b) {
    list lst;
    k = (a < b) ? -a : -b;
    a = a + b;
    lst[1] = k; lst[2] = a; lst[3] = b;
    return(lst);
 }

 lst = tff3(-15,27);
 print "List result=", lst;
 print " This should be 15: ",lst[1];
```

5.5.4 Data Lists and Structs

Lists and structs can be members of lists and structs:

```
struct str1;                                List1=
list lst1;
                                                  *************************
                                                  lst1 (List with 2 Entries)
str1.a = -15; str1.b = 27;                        *************************
lst1[1] = str1.a; lst1[2] = str1.b;
lst1[1] += lst1[1]; lst1[2] += lst1[2];
print "List1=", lst1;                                   lst1[1]:     -30
                                                        lst1[2]:      54
```

```
struct str1;
list lst1;

lst1[1] = [ 1 2 3 ]; lst1[2] = [ 3 2 1 ];
str1.a = lst1[1]; str1.b = lst1[2];
print "Struct1=", str1;
```

```
Struct1=

        *************************
        Struct str1 with 2 Entries
        *************************

      Struct str1 Entry[1]=str1.b:
      ***************************

        Dense Row Vector str1.b

        R |       1      2      3
                  3      2      1


      Struct str1 Entry[2]=str1.a:
      ***************************

        Dense Row Vector str1.a

        R |       1      2      3
                  1      2      3
```

5.6 Defining Structs

5.6.1 Syntax for Structs

Before using a struct, the name must be declared as a struct_name by the following keyword statement:

$$\texttt{struct}\ struct\_name <, struct\_name\_2, \ldots >;$$

Note, when the *struct_name* was used in CMAT code before this definition, the former object is freed by default and its content is lost, even if the name was already defined as a struct. Structs which are entries of structs do not have to be defined by the `struct` declaration.

Struct entries can be either scalars, vectors, matrices, tensors, lists, or structs.

After a name was defined as a struct, a struct entry can be defined in the notation:

$$struct\_name.entry\_name = name\_of\_data\_object$$

.

or

$$struct\_name.entry\_name[irange] = name\_of\_data\_object$$

.

Note, the *entry_name* and the *name_of_data_object* can be the same but must not necessarily be so. Struct entries can be used in arithmetic operations.

5.6.2 Simple Example for Using Structs

Here we assign a scalar `b` as an entry of struct `a`:

```
struct a;                          [1] Struct a=
a.b = 5;
print "[1] Struct a=", a;                 **************************
                                          Structure a with 1 Entries
                                          **************************

                                       Struct a Entry[1]=a.b:       5
```

This is how we access the entry `b` of struct `a`:

```
b = a.b;                           Entry b= 5
print "Entry b=",b;                 Struct Entry 2 * a.b= 10.000
a.b *= 2.;
print "Struct Entry 2 * a.b=",a.b;
```

Make matrix `c` to another entry of struct `a`:

```
c = [ 2. 1.,
      1. 2.];
a.c = c;
print "[2] Struct a=", a;
```

```
[2] Struct a=
      [1] Struct a with 2 Entries
      **************************

         Struct a Entry[1]=a.c:
         **********************

       Dense Symmetric Matrix a.c

        S |           1          2
        --------------------------
        1 |   2.0000000
        2 |   1.0000000  2.0000000

    Struct a Entry[2]=a.b:  10.0000000
```

After assigning struct `d` to an entry, the struct `a` now has three entries:

```
struct d;
d.ent = [ 1 2 3 ];
a.d = d;
print "[3] Struct a=", a;
lstmem(1);
lststk(1);
```

```
[3] Struct a=
      [2] Struct a with 3 Entries
      **************************

     [1] Struct a.d with 1 Entries
     *****************************

      Struct a.d Entry[1]=a.d.ent:
      ****************************

         Dense Row Vector d.ent

        R |       1      2      3
                  1      2      3

         Struct a Entry[2]=a.c:
         **********************

       Dense Symmetric Matrix a.c

        S |           1          2
        --------------------------
        1 |   2.0000000
        2 |   1.0000000  2.0000000

    Struct a Entry[3]=a.b:  10.0000000
```

Here we assign a vector `a.d.f` as a second entry of substruct `a.d`:

```
a.d.f = [ 3 2 1 ];
print "[4] Struct a=", a;
```

```
[4] Struct a=
      [3] Struct a with 3 Entries
      ***************************

     [1] Struct a.d with 2 Entries
     ****************************

       Struct a.d Entry[1]=a.d.f:
       *************************

         Dense Row Vector a.d.f

       R |       1      2      3
                 3      2      1


      Struct a.d Entry[2]=a.d.ent:
      ****************************

         Dense Row Vector d.ent

       R |       1      2      3
                 1      2      3


         Struct a Entry[2]=a.c:
         **********************

       Dense Symmetric Matrix a.c

       S |           1          2
       --------------------------
       1 |   2.0000000
       2 |   1.0000000  2.0000000


   Struct a Entry[3]=a.b:  10.0000000
```

```
print "[5] Struct a=", a;
```

```
[5] Struct a=
      [3] Struct a with 4 Entries
      **************************

     [1] Struct a.d with 2 Entries
     ****************************

       Struct a.d Entry[1]=a.d.f:
       *************************

         Dense Row Vector a.d.f

        R |       1      2      3
                  3      2      1

      Struct a.d Entry[2]=a.d.ent:
      ****************************

         Dense Row Vector d.ent

        R |       1      2      3
                  1      2      3

        Struct a Entry[2]=a.cb:
        ***********************

      Dense Symmetric Matrix a.cb

        S |           1          2
        --------------------------
        1 |   20.000000
        2 |   10.000000  20.000000


        Struct a Entry[3]=a.c:
        **********************

       Dense Symmetric Matrix a.c

        S |           1          2
        --------------------------
        1 |   2.0000000
        2 |   1.0000000  2.0000000

   Struct a Entry[4]=a.b:  10.0000000
```

The following show operations with struct entries:

```
cb =  a.c * a.b;
print "cb", cb;
a.cb = a.c * a.b;
print "Entry a.cb=", a.cb;
```

```
cb
 S |         1         2
-----------------------
 1 |    20.000
 2 |   10.0000    20.000

Entry a.cb=
 S |         1         2
-----------------------
 1 |    20.000
 2 |   10.0000    20.000
```

```
free a.d;
print "a after freeing a.d=", a;
```

```
a after freeing a.d=
      [4] Struct a with 3 Entries
      ***************************

        Struct a Entry[1]=a.cb:
        ***********************

      Dense Symmetric Matrix a.cb

        S |           1          2
        --------------------------
        1 |   20.000000
        2 |   10.000000  20.000000


         Struct a Entry[2]=a.c:
         **********************

       Dense Symmetric Matrix a.c

        S |           1          2
        --------------------------
        1 |   2.0000000
        2 |   1.0000000  2.0000000


   Struct a Entry[3]=a.b:  10.0000000
```

5.6.3 Structs and User Specified Functions

Structs can be arguments of user specified functions:

```
print " User Function with struct argument"; This should be 15:  15
 function tff2(str1) {
    a = str1.a; b = str1.b;
    k = (a < b) ? -a : -b;
    a = a + b;
    return(k);
 }

 struct str1;
 str1.a = -15; str1.b = 27;

 c = tff2(str1);
 print " This should be 15: ",c;
```

Structs can be returned by user specified functions:

```
print "User Function returns with Struct";   This should be 15:  15
 function tff4(a,b) {
    struct str;
    k = (a < b) ? -a : -b;
    a = a + b;
    str.k = k; str.a = a; str.b = b;
    return(str);
 }

 str = tff4(-15,27);
 lstvar(1);
 lstmem(1);
 print "Struct result=", str;
 print " This should be 15: ",str.k;
```

5.7 Keywords

The following keywords are used in CMAT and cannot be used as identifiers (names of variables, labels, or functions):

```
break     case      char    complex   conj    continue  default   else     exit     for
free    function  global  gnuplot   goto     gpend     herm      if      imag     int
link      list    option    poptn   print     psd      real    rename  return  sizeof
struct  switch    symm    tri2sym   void    while
```

List of Keywords

The keyword `option` has the alias `options`, and the keyword `poptn` has the alias `poptns`, which are also keywords and therefore, cannot be used for names of variables or functions.

5.8 Changing the Type of Variables (Casting)

The basic language elements of CMAT have been designed to be close to the C programming language, but with a few exceptions. Names of variables or functions (identifiers) must start with a letter and can be followed by at most 14 significant letters or digits. As with Unix, upper and lower case letters are treated differently. Each variable name, identifier, which is connected to a region of storage is called *lvalue* or *object* in C. The objects or lvalues in CMAT are scalars, vectors, or matrices.

The data type of a variable depends on its first evaluation. For example, `a = 1;` defines an `int` variable, `b = 2.;` defines a `real` variable, and `c=3-1i` defines a `complex` variable. Operations can change the data type. For example, the division of two integers results in a floating point number, the square root or logarithm of a negative `int` or `real` variable is a `complex` variable.

Variables can be *casted* to other data types using the type specification inside parentheses in front of the variable name, i.e. `(int)`, `(real)`, and `(complex)`, when a different data type is necessary for performing a specific operation (see [455], p. 42). Casting a missing value always results in a missing value. A matrix with missing values cannot be casted to `int`. For example, you can use the `(int)a` cast to convert the floating point value of variable a into an integer by chopping. Otherwise, you can use the `(real)` cast to convert the integer values of variables (scalars, vectors, or matrices) to floating point numbers to prevent the integer overflow of a succeeding arithmetic operation. Note that in the internal C implementation of CMAT, the `int` data type is treated as long, the `real` data type as double precision, and the `complex` data type in form of two double precision values.

```
z = .5-.8i;
y1 = (conj)z; y2 = (real)z - (imag)z*1.i;
print y1, y2;

y1= 0.5000 +0.8000i y2= 0.5000 +0.8000i
```

Since the most important part of CMAT is (hopefully, user friendly) matrix algebra, pointers are not needed in CMAT. For the one-dimensional case, the entries of a vector can be addressed in CMAT with indices written inside brackets `[` and `]` in the same way as one-dimensional arrays in C. In the k-dimensional case, $k \geq 2$, CMAT requires either k indices separated by commas or one single k-column matrix inside one pair of brackets.

Indices can be simple variables or subscript reduction operators (see below). Indices are usually one based like in FORTRAN, but using the option `INDBASE=0` or `IB=0`, you can easily change the execution to zero-based indices as in C. There is no significant difference in the computational performance between the two types of index base.

To illustrate zero base indices, an example similar to those of the previous section is selected,

```
a = [ 1 2,                          |   0   1   2   3
      3 4];                        --------------------
b = [ 5 6,                         0 |   1   2   0   0
      7 8];                        1 |   3   4   0   0
option indbase=0;                  2 |   0   0   5   6
a[2,2] = b;                        3 |   0   0   7   8
print a;
option indbase=1;
```

Cast	Variable	Description
(int)	real	The decimals are truncated, similar to the `fix()` function. If the value of the variable is larger than the largest long integer in computer arithmetic (MACLONG), the largest long integer is returned with the correct sign.
	complex	The imaginary part is ignored.
	char or string	The cast results in a missing value.
	matrix	The casting is performed elementwise. If the matrix contains missing values, a missing value is returned.
(real)	int	Type is changed to real.
	complex	The imaginary part is ignored.
	char or string	The cast results in a missing value.
	matrix	The casting is performed elementwise.
(complex)	int	Type is changed to real.
	real or complex	Result is unchanged.
	char or string	The cast results in a missing value.
	matrix	The casting is performed elementwise.
(conj)	int or real	Result is unchanged.
	complex	The conjugate complex variable $a - bi$ results.
	char or string	The cast results in a missing value.
	matrix	The casting is performed elementwise.
(imag)	int or real	Result is unchanged.
	complex	The imaginary part b of $z = a + bi$ is returned as `real` variable.
	char or string	The cast results in a missing value.
	matrix	The casting is performed elementwise.
(tri2sym)	scalar	Result is unchanged.
	matrix	The lower or upper triangular matrix $\mathbf{A}$ is changed to a symmetric matrix.
(symm)	scalar	Result is unchanged.
	matrix	The matrix $\mathbf{A}$ is symmetrized by its least-squares approximation $\hat{\mathbf{A}} = \frac{1}{2}(\mathbf{A} + \mathbf{A}^T)$.
(herm)	scalar	Result is unchanged.
	matrix	For complex matrices, the Hermitian matrix $\frac{1}{2}(\mathbf{A} + \mathbf{A}^H)$ is returned. For real or int matrices, the `(herm)` cast is identical to the `(symm)` cast.
(psd)	scalar	Result is unchanged.
	matrix	The matrix $\mathbf{A}$ is symmetrized to $\frac{1}{2}(\mathbf{A} + \mathbf{A}^T)$ and a specific bit is set in its attribute structure indicating that the matrix is positive semidefinite and is eligible for specific cost efficient operations like Cholesky factorization. Beware: Symmetrizing does not transform an indefinite matrix into a positive semidefinite matrix, it only ensures that later operations with this matrix will assume this property.

Casting Variables and Matrices

The casting to data type like `int`, `real`, `complex`, `conj`, and `imag` can be applied to tensors.

5.9 Arithmetic Operations

5.9.1 Scalar Arithmetic Operations

Unary Arithmetic Operations

Primary expressions are defined similar to C (see [455], p. 185) except that pointers are not permitted. The following unary operators are permitted:

Op	Name	Description
`-` *expression*	negative of expression	The type of the expression must be numeric (`int`, `real`, or `complex`). The result has the same data type.
`!` *expression*	logical negation	Results in one if the expression is equal to zero. Results in zero if the expression is not equal to zero. The expression may have any data type but the result is always `int`. If the expression is a string of zero length, negation results in one; if the string has nonzero length, negation results in zero.
`~`*expression*	one's complement	The type of the operand can be `int` or `real`. If it is `real`, the number must be representable as a long integer. The result is of type `int`.
`++` *lvalue*	prefix increment	The type of the operand must be numeric. The result of `++x` is the incremented value of x and the stored value of x is also incremented. This is equivalent to `x += 1`.
`--`*lvalue*	prefix decrement	Analogous to prefix increment. The result of `--x` is equivalent to `x -= 1`.
lvalue`++`	postfix increment	The type of the operand must be numeric. The result of `x++` is the value of x before the stored value of x is incremented.
lvalue`--`	postfix decrement	Analogous to postfix increment.

Unary Arithmetic Operators

Binary Arithmetic Operations

The following binary arithmetic operators are permitted:

Op	Name	Description
`*`	multiplication	The type of both operands must be numeric. The result has the dominant operand type. That means, if one of the two operands is `complex`, then the result is `complex`. Else if one of the two operands is `real`, then the result is `real`. Otherwise, the result is `int`.
`/`	forward division	`x / y` means that x is divided by y. The type of both operands must be numeric. The result is `complex`, if at least one operand is `complex`. Otherwise, the result is `real`.
\	backward division	`x \ y` means that y is divided by x. The type of both operands must be numeric. The result is `complex`, if at least one operand is `complex`. Otherwise, the result is `real`.
`%`	remainder int division	The type of both operands must be either `int` or `real`. A real operand must be castable to a long integer. The data type of the result is always `int`, the result has the same sign as the dividend. For $b \neq 0$ there is always `a == (a/b)*a + a%b`. For `real` results, see also `fmod(x,y)`.
`+`	addition	The type of both operands must be numeric. The result has the dominant operand type.
`-`	subtraction	The type of both operands must be numeric. The result is that of the dominant operand type.
`**`	power	The type of both operands must be numeric. The type of the result depends on the type of the two operands: If the left operand is not `complex` and the right operand is of type `int`, the power x^b is computed. The type of the left operand defines the type of the result. If the left operand is not `complex` and the right operand is of type `real` and nonnegative, the result x^y is computed using the power function `pow(x,y)` of the C compiler. If the left or right operand is of type `complex` or the right operand is negative `real`, the power function is extended for complex algebra.

Binary Arithmetic Operators

5.9.2 Shift, Relational, and Logical Operations

Shift, relational, and logical binary operations are permitted as in C. The operands of the following operations must be `int` or `real`. If one of the operands is `real` internally, the number is converted to a long integer. The result is always `int`.

Op	Name	Description
`<<`	left-shift operation	Like in [455], p. 189, the result is undefined if the right operand is negative, or greater than or equal to the length of the object in bits. The right operand specifies the number of bits by which the left operand (bit pattern) is shifted, the vacated bits are 0-filled. E.g. if i is an `int`, then `i << 1;` corresponds to `i*2`.
`>>`	right-shift operation	This is analogous to the left-shift operation. E.g. if i is an `int`, then `i >> 1;` corresponds to `i/2`.
`<`	less-than relation	The result is 1 if the specified relationship is true, otherwise the result is 0.
`>`	greater-than relation	The result is 1 if the specified relationship is true, otherwise the result is 0.
`<=`	less-than-or-equal-to relation	The result is 1 if the specified relationship is true, otherwise the result is 0.
`>=`	greater-than-or-equal-to relation	The result is 1 if the specified relationship is true, otherwise the result is 0.
`==`	equal-to relation	The result is 1 if the specified relationship is true, otherwise the result is 0. This operation is permitted for missing value(s).
`!=`	not-equal-to relation	The result is 1 if the specified relationship is true, otherwise the result is 0. This operation is permitted for missing value(s).
`&`	bitwise-and operation	The logical *and* operation is applied to corresponding bits of the two integer operands.
`|`	bitwise-inclusive-or operation	This is analogous to the bitwise-and operation.
`^`	bitwise-exclusive-or operation	This is analogous to the bitwise-and operation.
`&&`	logical-and operation	The result is 1 if both operands are non-zero, otherwise the result is 0.
`||`	logical-or operation	The result is 1 if at least one of the operands is non-zero, otherwise the result is 0.

Shift, Relational, and Logical Operators

x	y	x && y	x || y	x ^ y
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	1	0	0

Definition of Logical Operators

If the two operands are matrices (or vectors) they must have the same dimension. The result is a binary matrix (or vector) with the same size containing only zeros and one. If one of the operands is a scalar and the other is a matrix (or vector) then the result is a binary matrix (or vector) resulting from pairwise comparisons among each entry of the matrix with the same scalar.

5.9.3 Extension of Scalar Operations to Matrix Arithmetic

Unary Operations

- If the operand x of the unary operation is a vector or matrix, then the operation is performed on each entry of the object. If the object contains string data or missing values, then the operations `-`, `~`, `++`, and `--` return missing values.
- There are two more unary operations for matrices $\mathbf{A} = (a_{ij})$: The ‘ operation defines the *conjugate transpose* $\mathbf{A}^H = (\bar{a_{ji}})$ and the operation .‘ defines the *transpose* $\mathbf{A}^T = (a_{ji})$ of $\mathbf{A} = (a_{ij})$. If $\mathbf{A} = (a_{ij})$ does not contain complex data then both transposes are the same.

Op	Name
‘	transpose
.‘	nonconjugate transpose

Additional Unary Operators

Most unary matrix operations can be applied to tensors in a trivial way. A generalization of the transpose for matrices is provided by the `b = tenperm(a,ordr)` function which permutes the dimensions of an input tensor`a` in a specified order `ordr`.

Binary Operations

- The Kronecker product `@` cannot be applied to tensors.
- The power operation `**` is extended in various ways:
  - The left operand can be a square $n \times n$ matrix $\mathbf{A}$ and the right operand b an `int` or `real` scalar.
    - If the right operand is an `int` scalar b, the matrix power $\mathbf{A}^b$ is computed using an algorithm of Knuth [462]. If b is negative, the matrix power $(\mathbf{A}^{-1})^{-b}$ is computed based on the inverse of $\mathbf{A}$.
    - If the right operand is a `real` scalar y, the matrix power $\mathbf{A}^y$ is computed using the eigenvalue decomposition of $\mathbf{A}$.

    This power operation cannot be applied to a tensor $\mathbf{A}$.
  - The left operand `a` can be a scalar and the right operand `b` can be an $n \times m$ matrix or vector $\mathbf{B}$. Then the result is an $n \times m$ matrix containing the values $a^{b_{ij}}$ for each entry of $\mathbf{B}$. This elementwise power operation can also be applied to tensors.
  - The left operand and the right operand can be matrices, or vectors $\mathbf{A}$ and $\mathbf{B}$ of the same dimensions, then the power $a_{ij}^{b_{ij}}$ is computed between corresponding entries of $\mathbf{A}$ and $\mathbf{B}$. This elementwise power operation can also be applied to tensors.
- Compatibility of matrix and vector multiplication with $*$: If both operands $\mathbf{A}$ and $\mathbf{B}$ of the `*` operation are vectors or matrices, then the common matrix multiplication $\mathbf{A} * \mathbf{B}$ is performed. The dimensions of the two operands must be compatible, that means for `A * B` the number of columns of $\mathbf{A}$ must be equal to the number of rows of $\mathbf{B}$. The result has the number of rows of $\mathbf{A}$ and the number of columns of $\mathbf{B}$.
- Compatibility of tensor multiplication with $*$: If a tensor $\mathbf{A}$ is multiplied with a matrix or vector $\mathbf{B}$, then the size of the last dimension of the tensor $\mathbf{A}$ must be the same as the number of rows of $\mathbf{B}$. If a matrix or vector $\mathbf{A}$ is multiplied with a tensor $\mathbf{B}$, then the number of columns of $\mathbf{A}$ must be the same as the size of the first dimension of the tensor $\mathbf{B}$. For $\mathbf{C} = \mathbf{A} * \mathbf{B}$, where both, $\mathbf{A}$ and $\mathbf{B}$ are tensors the dimensionsionality d of $\mathbf{C}$ is:

  - $d = 0$ i.e. $\mathbf{C}$ is scalar: when $\mathbf{A}$ and $\mathbf{B}$ have the size.
  - $d = d_1 + d_2$: if d_A and d_B are the dimensionalities of the tensors $\mathbf{A}$ and $\mathbf{B}$, and s is the number of first dimension with equal size, then $d_1 = d_A - s$ and $d_2 = d_B - s$.

  The functions `tentvec`, `tentmat`, and `tentten` can be used for more specific forms of tensor multiplication.

- The operations `/` and \ cannot be applied for tensors. However, the elementwise operations `./` and `.\` can be applied to tensors of equal size.

- If both operands $\mathbf{A}$ and $\mathbf{B}$ of the `/` and \ operation matrices, the division is replaced by the solution(s) of linear system(s):

  `x= a / b` forward division: $\mathbf{X} = \mathbf{B}^{-1}\mathbf{A}$

  `y= a \ b` backward division: $\mathbf{Y} = \mathbf{A}^{-1}\mathbf{B}$

  which can be mathematically (but not numerically) understood as the common matrix multiplication combined with matrix inversion. The linear system is solved numerically by matrix decomposition (factorization) followed by the solution of linear systems with the simpler factor matrices. The dimensions of the matrices $\mathbf{A}$ and $\mathbf{B}$ must be compatible. That means, the matrix to be inverted must be square $n \times n$ and the corresponding dimension of the other matrix must be compatible for the product.

- Except for the power operation `**`: If one of the two operands is scalar and the other operand is a tensor, matrix, or vector, then the operation is performed between the scalar and each entry of the other operand.

- Except for multiplication `*`, forward division `/`, and backward division \: If both operands are tensors, matrices, or vectors, then both must have the same dimensions (for matrices the same number of rows and columns). The operation is then done elementwise between corresponding entries of the two operands.

- An operation results in a missing value, if one or both operands of an operation is string or missing value.

Additional Binary Operations

For the following operations, except the Kronecker product `@`:

1. Both operands $\mathbf{A}$ and $\mathbf{B}$ must have the same dimension (same number of rows and columns).
2. The operation are performed between corresponding elements of $\mathbf{A}$ and $\mathbf{B}$.

Op	Name
`.*`	elementwise multiplication
`./`	elementwise forward division
`.\`	elementwise backward division
`.**`	elementwise power
`@`	Kronecker product (see below)
`.<`	elementwise less-than relation
`.>`	elementwise greater-than relation
`.<=`	elementwise less-than-or-equal-to relation
`.>=`	elementwise greater-than-or-equal-to relation
`.==`	elementwise equal-to relation
`.!=`	elementwise not-equal-to relation

Note: as with the `==` and `!=` operations, the operations `.==` and `.!=` are permitted for missing value(s).

Additional Binary Operators

Example for Kronecker product: If **A** is an $n \times m$ matrix and **B** is an $p \times q$ matrix then **C** from `c = a @ b;` is the $np \times mq$ matrix of all products $a_{ij}b_{kl}$.

```
a = [ 1 2,
      3 4 ];
b = [ 3 4,
      4 5 ];
c = a @ b;
print c;
```

```
   |    1    2    3    4
---------------------------
 1 |    3    4    6    8
 2 |    4    5    8   10
 3 |    9   12   12   16
 4 |   12   15   16   20
```

When applied to vectors or matrices, the common relational operators `<, >, <=, >=, ==`, and `!=` return a binary scalar 1 or 0 depending on the fact whether the relationship is satisfied for **all** pairs of elements or not. The elementwise relational operators `.<, .>, .<=, .>=, .==`, and `.!=` return either a binary object of the same dimension or a scalar depending on:

- If both operands have the same dimension or one of the operands is a scalar, the comparison is performed for each corresponding pair of elements and a vector or matrix of the same dimension is returned containing only ones or zeros indicating the result of each comparison.
- Otherwise the scalar 0 is returned.

```
a = [ 1 2 3,
      4 5 6 ];
b = [ 3 2 1,
      6 5 4 ];
c = a .> b; d = a .> 3;
print c, d;
```

```
  U |   1   2   3
---------------
  1 |   0   0   1
  2 |       0   1

    |   1   2   3
---------------
  1 |   0   0   0
  2 |   1   1   1
```

5.10 Additional Operators

5.10.1 Concatenation Operators

The following concatenation operations between vectors and matrices were already documented:

`->`	horizontal concatenation	from left to right
`<-`	horizontal concatenation	from right to left
`|>`	vertical concatenation	from top to bottom
`<|`	vertical concatenation	from bottom to top
`\>`	diagonal concatenation	from left top to right bottom
`</`	diagonal concatenation	from left bottom to right top

Concatenation Operators

The result of a matrix concatenation has the dominant data type of the two operands.

5.10.2 Conditional Operator

The syntax is the same as in C:

expression `?` *expression* `:` *expression*

The result depends on the evaluation of the left operand. If the left operand result is non-zero, the result is the middle operand, otherwise the result is given by the last operand. For example, the following code is compatible with C,

```
b = 5; c = 3;
a = sqrt((c + 3 > b) ? c : b);
print " a*a should be equal to c=3:", a * a;
```

5.10.3 Assignment Operators

Each assignment, *lvalue = expression* connects the result of the right-hand side expression with the storage area of the left lvalue. Different from C, the data type of the lvalue is defined by the type of the result of the expression on the right.

CMAT (like C, see [455], p.191) permits the use of the following compound assignment operators:

Op	Description	a	b	a Op b
`+=`	addition	3.5	2.5	6.0
`-=`	subtraction	3.5	2.5	1.0
`*=`	multiplication	3.5	2.5	8.75
`/=`	forward division	6	3	2.
`\ =`	backward division	6	3	.5
`%=`	remainder	5	3	2
`@=`	Kronecker product	`[1 2]`	$\begin{bmatrix} 2 \\ 3 \end{bmatrix}$	$\begin{bmatrix} 2 & 4 \\ 3 & 6 \end{bmatrix}$
`<<=`	left-shift	4	2	16
`>>=`	right-shift	4	2	1
`&=`	bitwise-and	6	5	4
`|=`	bitwise-or	6	5	7
`^=`	bitwise-exclusive-or	6	5	1

Compound Assignment Operators

Like in C, expressions can be separated by comma. For example, you may program like in [455], p.192:

```
b = 2;
a = atan2(b, (t = 4, t += 2));
print " atan2(2,6): ", a, " == ", atan2(2,6);
```

5.10.4 Subscript Reduction Operators

You can work with zero or one-based subscripts in CMAT, depending on the value of the `INDBASE=` option. If not otherwise stated, we assume in this report that the subscripts are one-based like in Fortran. The entries of vectors or matrices are referred to by integer subscript expressions in brackets `[` and `]`, for example, `a[3]` refers to the entry of a row or column vector a with subscript 3, and `b[2,4]` refers to the element in the second row and fourth column of matrix **B**. If the subscript expression is replaced by one of the following subscript reduction operators then an operation is performed which reduces a vector to a scalar and a matrix to a vector. The following table contains examples of subscript reduction operations performed on the matrix:

```
a = [1 2 3 4 5,
     2 3 4 5 6,
     3 4 5 6 7];
print a[,+], a[+,+], a[+,*], a[**,];
print a[,<>], a[><,], a[><,<>], a[,<:>], a[,>:<,];
print a[<|>,], a[,>|<], a[<!>,], a[,>!<];
```

Note that all subscript reduction operators can be applied to tensors.

Op	Description	Example	Result
+	sum	a[,+]	[15, 20, 25]
		a[+,+]	60
*	product	a[+,*]	3360
|+|	sum of absolute values	a[,|+|]	[15, 20, 25]
|*|	product of abs. values	a[+,|*|]	3360
**	sum-of-squares	a[**,]	[14 29 50 77 110]
<>	maximum value	a[,<>]	[5, 6, 7]
><	minimum value	a[><,]	[1 2 3 4 5]
		a[><,<>]	5
<:>	index of maximum value	a[,<:>]	[5, 5, 5]
>:<	index of minimum value	a[>:<,]	[1 1 1 1 1]
<|>	maximum absolute value	a[<|>,]	[3 4 5 6 7]
>|<	minimum absolute value	a[,>|<]	[1, 2, 3]
<!>	index of max. abs. value	a[<!>,]	[3 3 3 3 3]
>!<	index of min. abs. value	a[,>!<]	[1, 1, 1]

Subscript Reduction Operators

Note, that subscript reduction operators applied to missing values yield missing values. You could use the `loc()` function with the `"nm"` option for locating the missing values before applying the subscript reduction operator.

5.10.5 Sorting and Ranking Operators

The elements of vectors or matrices, as well as selected rows and columns of matrices can be sorted using a set of index operators very similar to that of the subscript reduction operators. The operations also permit to return the ranks of the elements. The ranks may be zero or one based depending on the specification of the `INDBASE=` option. The ordering or ranking can be placed in each index location. For example, for the $m \times n$ matrix $\mathbf{A}$, the *order ascending* operator $<$ can be used in following ways:

`B = A[ind,<];` where `ind` is a k index vector containing indices specifying rows of matrix $\mathbf{A}$: the result is a $k \times n$ matrix $\mathbf{B}$ containing the selected rows of $\mathbf{A}$ sorted in ascending order.

`B = A[.,<];` a missing value or empty row (column) index location results in an $m \times n$ matrix $\mathbf{B}$ where each row of $\mathbf{B}$ contains the corresponding row of $\mathbf{A}$ sorted in ascending order.

`B = A[<,<];` the same sorting operator in both index locations results in an $m \times n$ matrix $\mathbf{B}$ which contains all the elements of $\mathbf{A}$ sorted in ascending order. (The content of $\mathbf{A}$ is treated as a single vector in the sorting process.)

The following example illustrates the operators when applied to the first row of the matrix $\mathbf{A}$:

```
a = [ 3 -4  5 -1  2,
     -4  5 -6  2 -3,
      5 -6 -7 -4  5];
print a[1,<],  a[1,>],  a[1,<:], a[1,>:];
print a[1,<|], a[1,>|], a[1,<!], a[1,>!];
```

Op	Description	Example	Result
<	ascending values	a[1,<]	[-4, -1, 2, 3, 5]
>	descending values	a[1,>]	[5, 3, 2, -1, -4]
<:	ranks of ascending values	a[1,<:]	[2, 4, 5, 1, 3]
>:	ranks of descending values	a[1,>:]	[3, 1, 5, 4, 2]
<|	ascending absolute values	a[1,<|]	[1, 2, 3, 4, 5]
>|	descending absolute values	a[1,>|]	[5, 4, 3, 2, 1]
<!	ranks of asc. abs. values	a[1,<!]	[4, 5, 1, 2, 3]
>!	ranks of desc. abs. values	a[1,>!]	[3, 2, 1, 5, 4]

Sorting and Ranking Operators

The ranking operation is especially important for obtaining index vectors needed for rearranging sets of rows or columns with respect to the same order.

```
a = [ 3 -4  5 -1  2,
     -4  5 -6  2 -3,
      5 -6 -7 -4  5];
ind = a[1,<:]; b = a[,ind];
print "Rows ordered w.r.t. ranks of first row:",b;
```

```
Rows ordered w.r.t. ranks of first row:
   |   1   2   3   4   5
-----------------------
 1 |  -4  -1   2   3   5
 2 |   5   2  -3  -4  -6
 3 |  -6  -4   5   5  -7
```

```
print "Sort/Rank each row separately",
      a[ ,<],  a[ ,>],  a[ ,<:], a[ ,>:],
      a[ ,<|], a[ ,>|], a[ ,<!], a[ ,>!];
```

```
   |   1   2   3   4   5
---------------------------
 1 |  -4  -1   2   3   5
 2 |  -6  -4  -3   2   5
 3 |  -7  -6  -4   5   5
```

```
   |   1   2   3   4   5
---------------------------
 1 |   1   2   3   4   5
 2 |   2   3   4   5   6
 3 |   4   5   5   6   7
```

```
   |   1   2   3   4   5
---------------------------
 1 |   5   3   2  -1  -4
 2 |   5   2  -3  -4  -6
 3 |   5   5  -4  -6  -7
```

```
   |   1   2   3   4   5
---------------------------
 1 |   5   4   3   2   1
 2 |   6   5   4   3   2
 3 |   7   6   5   5   4
```

```
   |   1   2   3   4   5
---------------------------
 1 |   2   4   5   1   3
 2 |   3   1   5   4   2
 3 |   3   2   4   1   5
```

```
   |   1   2   3   4   5
---------------------------
 1 |   4   5   1   2   3
 2 |   4   5   1   2   3
 3 |   4   1   5   2   3
```

```
   |   1   2   3   4   5
---------------------------
 1 |   3   1   5   4   2
 2 |   2   4   5   1   3
 3 |   1   5   4   2   3
```

```
   |   1   2   3   4   5
---------------------------
 1 |   3   2   1   5   4
 2 |   3   2   1   5   4
 3 |   3   2   1   5   4
```

```
print "Sort/Rank entire matrix as one vector",
      a[<,<],   a[>,>],   a[<:,<:], a[>:,>:],
      a[<|,<|], a[>|,>|], a[<!,<!], a[>!,>!];
```

```
   |   1   2   3   4   5
---------------------------
 1 |  -7  -6  -6  -4  -4
 2 |  -4  -3  -1   2   2
 3 |   3   5   5   5   5
```

```
   |   1   2   3   4   5
---------------------------
 1 |   1   2   2   3   3
 2 |   4   4   4   5   5
 3 |   5   5   6   6   7
```

```
   |   1   2   3   4   5
---------------------------
 1 |   5   5   5   5   3
 2 |   2   2  -1  -3  -4
 3 |  -4  -4  -6  -6  -7
```

```
   |   1   2   3   4   5
---------------------------
 1 |   7   6   6   5   5
 2 |   5   5   4   4   4
 3 |   3   3   2   2   1
```

```
   |    1    2    3    4    5
-----------------------------
 1 |   13    8   12   14    2
 2 |    6   10    4    9    5
 3 |    1   11    3    7   15
```

```
   |    1    2    3    4    5
-----------------------------
 1 |    4    9    5   10    1
 2 |    6   14    2   15    3
 3 |   11    7   12    8   13
```

```
   |    1    2    3    4    5
-----------------------------
 1 |   15    3   11    7    1
 2 |    9    5    4   10    6
 3 |    2   14   12    8   13
```

```
   |    1    2    3    4    5
-----------------------------
 1 |   13    8   12   11   15
 2 |    3    7    2    6   14
 3 |    1   10    9    5    4
```

The ~ index operator is very similar to the sorting and ranking operators. It may be used for randomly permuting the rows and/or columns of a vector or matrix. Using the ~ operator for the row (column) index will permute all specified columns (rows) with the same permutation. Using the ~ operator for both index positions yields different results in the following two cases:

1. For symmetric and diagonal matrices the same random permutation is used for rows and columns.
2. For all other matrix types different random permutations are used for rows than for columns.

The following example shows the unsymmetric case:

```
a = [1 2 3 ,
     4 5 6 ,
     7 8 9 ];
srand(1);
b1 = a[~,.];
b2 = a[.,~];
b3 = a[~,~];
print b1, b2, b3;
```

```
    |   1   2   3
----------------
 1 |   1   2   3
 2 |   7   8   9
 3 |   4   5   6

    |   1   2   3
----------------
 1 |   2   1   3
 2 |   5   4   6
 3 |   8   7   9

    |   1   2   3
----------------
 1 |   6   4   5
 2 |   9   7   8
 3 |   3   1   2
```

The following example shows the symmetric case. In general, only the double ~ operator yields a symmetric result:

```
a2 = [ 1 2 3 ,
       2 5 4 ,
       3 4 6 ];
srand(1);
b1 = a2[~,.];
b2 = a2[.,~];
b3 = a2[~,~];
print b1, b2, b3;
```

```
    |   1   2   3
----------------
 1 |   1   2   3
 2 |   3   4   6
 3 |   2   5   4

    |   1   2   3
----------------
 1 |   2   1   3
 2 |   5   2   4
 3 |   4   3   6

 S |   1   2   3
----------------
 1 |   6
 2 |   3   4
 3 |   3   1   2
```

If it is important to know which permutation was used, it may be recovered by using the `randperm()` function applying the same seed argument for the random generator:

```
a = [ 1  0  0  2  0  0  3  0 ,
      0  2  0  1  0  0  0  0 ,
      0  0  3  0  1  0  1  0 ,
      2  1  0  4  0  0  0  0 ,
      0  0  1  0  5  0  0  0 ,
      0  0  0  0  0  6  0  0 ,
      3  0  1  0  0  0  7  0 ,
      0  0  0  0  0  0  0  8 ];

srand(1);
b1 = a[~,~];
print b1;
```

```
srand(1);
ind = randperm(8);
print "This is the permutation used", ind;
b2 = a[ind,ind];
print "Should be zero:", ssq(b1-b2);
```

```
 S |   1   2   3   4   5   6   7   8
-----------------------------------
 1 |   2
 2 |   0   8
 3 |   0   0   5
 4 |   0   0   1   3
 5 |   0   0   0   0   1
 6 |   1   0   0   0   2   4
 7 |   0   0   0   0   0   0   6
 8 |   0   0   0   1   3   0   0   7
```

```
   |   1
--------
 1 |   2
 2 |   8
 3 |   5
 4 |   3
 5 |   1
 6 |   4
 7 |   6
 8 |   7

Should be zero: 0.0000
```

5.10.6 Precedence and Associativity of Operators

The following table shows the precedence of operators in CMAT and looks very similar to the one in [455], p. 49:

Rank	Operator	Associativity
1	`( ) [ ] { } < >`	left to right
2	`++ -- ! ~` sign: `+ -`	right to left
3	`** .** ‘ .‘`	left to right
4	`* / \ % @ .* ./ .\`	left to right
	subred: `+ * |+| |*| **`	left to right
	`<> >< <:> >:< <|> >|< <!> >!<`	left to right
	sort: `< > <: >: <| >| <! >!`	left to right
5	concat: `-> <- |> <| \> </`	left to right
6	op: `+ -`	left to right
7	`<< >>`	left to right
8	`< > <= >= .< .> .<= .>=`	left to right
9	`== != .== .!=`	left to right
10	`&`	left to right
11	`^`	left to right
12	`|`	left to right
13	`&&`	left to right
14	`||`	left to right
15	`? :`	right to left
16	`= += -= *= /= \ = %= @=`	right to left
	`&= |= ^= >>= <<=`	right to left
17	,	left to right

Precedence and Associativity of Operators

5.11 Statements

The following set of statements is available in CMAT. Most of the statements are very similar to those used in C (see [455], p. 201-204)

- **Compound and Null Statements**:
  Syntax: { *list_of_statements* }
  A block of statements combined inside of { and } braces is called a *compound statement*. Compound statements are often used in `if`, `while`, and `for` statements. The *null* statement consists only of a semicolon. When a label is being placed in front of a closing brace } of a compound statement a null statement is necessary as in C programming.

- **Break Statement**:
  Syntax: `break;`
  The `break` statement terminates executing inside the (compound) statement belonging to the nearest enclosing `while`, `for`, or `switch` statement. It continues execution at the statement following the terminated statement.

- **Continue Statement**:
  Syntax: `continue;`
  As you program in C (see [455], p. 203) the `continue` statement interrupts code execution within the compound statement belonging to a `while` or a `for` statement and continues execution at the end of the nearest enclosing `while` or `for` statement.

- **Rename Statement**:
  Syntax: `rename vnew = vold;`
  The `rename` statement with the following syntax `rename` *vnew* `=` *vold*`;` assigns a new name *vnew* to an existing variable with the old name *vold*. Whenever a variable with name *vold* does not exist or a variable with name *vnew* exists already, an error message is printed into the log and there will be no change of the variable structures. Note, that `rename` is a token and that this string cannot be used for variable or function name.

- **Exit Statement**:
  Syntax: `exit;` The `exit` statement can be used to terminate the execution of CMAT on any location of the input program. For example, it is useful in batch jobs to check the proper return of functions and interrupt the execution of the job if a crucial result was set to a missing value due to program or data errors.

- **For Statement**:
  Syntax: `for (` *opt_exp_1; opt_exp_2; opt_exp_3* `)` *statement*
  Like in C (see [455], p. 202) the `for` statement is equivalent to:

  ```
  opt_exp_1;
  while ( opt_exp_2 ) {
          ... statement ...
          opt_exp_3;
  }
  ```

  Each of the three expressions is optional. The first expression (*opt_exp_1;*) is the loop initialization, the second expression (*opt_exp_2*) specifies a test for the loop termination, and the third expression (*opt_exp_3;*) specifies an update of the loop variable which is usually an incrementation. If the second expression is missing, the `while` clause is by default `while(1)`. When CMAT is run in interactive mode, the input of the entire `for` statement must be completed (compiled) before it can be executed.

- **Free Statement**:
  Syntax: free *identifier* ... *identifier*;
  A `free` statement can be used to release memory for vectors or matrices. Note that the result of the statement `B[irb,icb] = C;` with missing value indices depends on the fact whether `B` is preset or freed.

- **Goto Statement and Labels**:
  Syntax: goto *identifier*;
  As it is used in C, a label is an identifier followed by a colon. Statements can be labeled in CMAT. The `goto` statement redirects the continued execution to the statement following the matching label. The label identifiers must be unique inside the program body or inside each user specified function. Note, that a label must be followed by a statement. For example, the label cannot be in front of a closing brace, }, of a compound statement. In this case the label should be followed by a *null statement* which consists only of a colon, `;`. In CMAT program code, different from C, we make a distinction between backward and forward `goto` statement. The `goto` statement is called a backward `goto` if it refers to a label connected to a statement preceeding it, and it is called a forward `goto` if it refers to a label connected to a statement afterward. When CMAT is run as an interpreter, a backward `goto` is immediately executed to already compiled assembler code. A forward `goto` postpones execution and only compiles the CMAT statements until it finds the matching target label.

- **If (Conditional) Statement**:
  Syntax: As in C (see [455], p. 201) there are two forms of conditional statements:

  - `if (` *expresssion* `)` *statement*
  - `if (` *expresssion* `)` *statement* `else` *statement*

  If the result of the expression in parenthesis is non-zero, then the first statement is executed. If the result is zero, in the first case the next statement is executed, in the second case the statement after the `else` is executed. When CMAT is run as an interpreter the entire conditional statement must be compiled before it can be executed. If there are more than one `if` statements in front of an `else`, the `else` refers to the last `if` statement just in front of the `else`. When CMAT is run in interactive mode, the input of the entire `if` statement must be completed (compiled) before it can be executed. Since the compiler always expects an `else`, the execution of the *if* statement is delayed until the input of an additional statement is completed. (You can use an empty statement `;` to force the execution.)

- **Link Statement**:
  Syntax: `link` *identifier* `;`
  A `link` statement is similar to a `goto` statement. The normal code execution is interrupted and continued at the label that matches the identifier. Differing from the `goto` statement, the `link` statement must have a `return` statement following it. This `return` statement returns the code execution to the statement after the `link` statement. Like the `goto` statement, a forward `link` postpones execution and only compiles the CMAT statements until it finds the matching target label.

- **Option Statement**:
  Syntax: `option` *optn* ... *optn* `;`
  where *optn* := *option_keyword* < =*option_setting* >
  The `option` statement permits the user to specify some useful runtime options like linesize (number of characters fitting on one line), pagesize (number of lines fitting on one page), the indexbase (usually 0 for C style or 1 for FORTRAN style), the optimal blocksize for blockwise LAPACK subroutines, and many more by using the `option` statement. See page 98 for a list of currently available options.

- **Print Statement**:
  Syntax: `print` < `poptn` ... `:` > *expression* `,` ... `,` < `poptn` ... `:` > *expression* `;`
  The print statement lists a number of expressions (returning a scalar, vector, or matrix result) **separated**

by comma. Each expression may be preceded by a `poptn ... :` clause specifying output options concerning only the result of the following expression. See page 101 for a list of currently available print options. The output is directed to standard output or to the file listed in the last `txtfile()` function call. To print or format scalar output you may also use the C compatible `printf()` and `sprintf()` functions.

- **Return Statement**:
  Syntax: `return;` or: `return` *expression* `;`
  A `return` statement performs like a `goto` statement. It interrupts and continues the execution at a statement which is predefined in one of the following two ways:
  - If the `return` statement corresponds to a `link` statement, the execution is continued at the statement following the `link`. Only the first syntax form applies.
  - Otherwise the `return` statement corresponds to a function call. The first syntax form does not return a value, the second form returns a function output value to the call.

- **Switch Statement**:
  Syntax: `switch (` *expression* `)` *compound statement*
  The result of the expression must be `int`. Since in CMAT `char` variables (single quotes) are treated like strings (double quotes), `char` variables cannot be used in `switch` statements. The compound statement contains a list of statement blocks which are labeled either with `case` *constant*`:` or `default :`. The constant can be either `int` or `char`. No more than one `default :` label can be used. After evaluating the integer expression, executing is continued at a matching `case` label and if no matching `case` label is found, executing is continued at the `default` label. If no `case` matches and there is no `default` label then none of the statements inside the switch statement is executed. The `break` statement can be used to continue processing after the `switch` statement (see [455], p. 203). When CMAT is run in interactive mode, the input of the entire `switch` statement must be completed (compiled) before it can be executed.

- **While Statement**:
  Syntax: `while (` *expresssion* `)` *statement*
  The statement at the end is executed repeatedly as long as the value of the expression in parenthesis remains non-zero. When CMAT is run in interactive mode, the input of the entire `while` statement must be completed (compiled) before it can be executed.

- **Calling Standard Functions**:
  Most standard functions return at least one result `x` (scalar, vector, or matrix) depending on various input arguments `a, b,...`. In these cases the function call can be a single statement `x = fname(a,b,..);` or can be used directly in arithmetic expressions, for example, `y = c * sqrt(a) / b;` or as arguments in function calls. Several standard functions may have the ability to return more than one result (e.g. x, y, z). The syntax for calling such functions is as follows: `< x, y, z > = fname(a,b,c);` . Whereas the order of the return values is important, any of the result names may be missing as in the following example, `< , , z > = fname(a,b,c);` where only result `z` is returned. For some of the standard functions some of the input arguments have default values and do not need to be specified. In these cases a missing value (period) can be used to skip across the input argument, e.g. `a = cons(3,.,'d',5);`.

5.12 User Defined Functions

1. < *type_def* > `function` *f_name* `(` *opt_parm_list* `)` *compound_statement*
2. < *type_def* > `function` *f_name* `(` *opt_parm_list* `)` `global` `(` *glob_parm_list* `)` *compound_statement*

where

- < *type_def* > is optional and may be one of the following, `void`, `int`, `real`, `complex`, or `char`,
- *opt_parm_list* is either empty or consists of a list of formal variable or function names separated by comma,
- *glob_parm_list* consists of a list of formal variable names separated by comma,
- names of function parameters must be followed by a pair of parentheses`()`.

The syntax form 1 is similar to that used in C language for function definitions, except for the additional `function` and `global` keywords. The actual function call does not contain the `global` keyword and its list of global parameters.

1. For the first list (*opt_parm_list*) the actual arguments are copied from their global into local locations, like in C. The actual arguments of the *opt_parm_list* must be initialized before the function is called. They may be modified inside the function but the modification is not returned into the global locations. The actual arguments may be terms of veriables like `(a+b)*exp(c)`.
2. If the function definition contains the `global` keyword with a list of globally defined variable names, then its content is **not** copied to local variables, i.e. changing any of these variables inside the function will change its global value.
3. The type definition of a function definition is optional. The results are normally returned with the types to which they are set during the execution of the function. If the type definition `type_def` is specified it has the effect of casting the type of the first result.
4. If function definitions are nested, then the global arguments of the inner functions must be also global arguments of the outer functions.

If the user specified function `fname(d,e,...,f)` should return more than one result, its function definition must contain a `return` statement of the form `return(a,b,...,c);` which lists the $m > 1$ return arguments which then can be referred to in calls of the form `<a,b,..c> = fname(d,e,...,f);`. In this case a type definition of the function refers only to the first return argument.

The following function definition uses global parameters `h` and `pi` which must be initialized before `fx0` is called:

```
function fx0(x) global(h,pi) {
   var = h * h + 1.;
   y  = (x - 2.) * exp( ((x - 2.) * (x - 2.)) / (-2. * var) );
   y += (x + 2.) * exp( ((x + 2.) * (x + 2.)) / (-2. * var) );
   y /= ( 2. * sqrt(2. * pi * var) * var );
   /* printf("\n x= %16.6e, fx= %16.6e\n",x,y); */
   return(-y);
}
```

Now h and π are defined globally before the function `fx0` is called without listing the global parameters:

```
str = "Miguel's Function:";
ab = [ 0., 6. ]; x0 = ab[2]; h = 0.;
pi = macon("pi");
y = fx0(x0);
print str," x,f(x)=",x0, y;
```

The advantage of the `global` specification is best illustrated when the function `fx0` is used as input argument for `fzero()` since `fzero` will call `fx(x)` for different values of x but does not know about the global variables required by `fx0`:

```
str = "Miguel's Function:";
ab = [ 0., 6. ]; x0 = ab[1]; h = 0.;
pi = macon("pi");
z1 = fzero(fx0,x0,ab,"bre");
print str," Brent=", z1, fx0(z1);
```

When modified inside the function, the global variables can be considered as additional return values.

Since CMAT can be executed interactively, a function must be defined before it can be called. This is different from C where a function can be defined anywhere (because in C the code is first compiled and then linked before it is executed). Also different from C: in CMAT functions cannot be nested. You must use the keyword `function` to define a user specified function which can be preceded by a data type (`int`, `real`, or `complex`), similar to Fortran syntax. For example, the following function returns the real square root of an absolute valued argument,

```
function rsqrt(a) {
  b = (a < 0) ? -a : a;
  return(sqrt(b)); }
print "rsqrt=", rsqrt(-9), " should be 3.\n";
```

In CMAT (as in C) you can code recursive functions like the following Euclid's algorithm to find the greatest common divisor of two numbers:

```
int function gcd(a,b) {
  if (a == b) return a;
  else if (a > b) return(gcd(a-b,b));
             else return(gcd(a,b-a));
}
a = 36; b = 54;
c = gcd(a,b);
print "\n Euclid=", c, " should be 18";
```

The following are two examples for multiple return functions:

```
function funone(d,e,f) {
  c = d + e + f;
  return c;
}

d = 3; e = 5.5; f = 3.1;
z = funone(d,e,f);
print "This is 11.6:", z;

function funtwo(g,h,i)  {
   /* g=3,h=1,i=2: d=6 */
   d = funone(g,h,i);
   a = d + d;
   b = h * h;
   c = g + h + i;
   /* a=12, b=1, c=6 */
   return(a,b,c);
}

g = 3; h = 1; i = 2;
< a,b,c > = funtwo(g,h,i);
/* this should be: a=12, b=1, c=6 */
print "Results:", a, b, c;
```

```
function funmor(d,e,f)  {
   a = d + d;
   b = e * e;
   c = d + e + f;
   return(a,b,c);
}

d = 3; e = 5.5; f = 3.1;
< a,b,c > = funmor(d,e,f);
print "Results 1:", a, b, c;

function funtwo(g,h,i)  {
   /* g=3,h=1,i=2: d=6, e=1, f=6 */
   < d,e,f > = funmor(g,h,i);
   a = d + d;
   b = e * e;
   c = d + e + f;
   /* a=12, b=1, c=13 */
   return(a,b,c);
}

g = 3; h = 1; i = 2;
/* this should be: a=12,b=1,c=13 */
< a,b,c > = funtwo(g,h,i);
print "Results 2:", a, b, c;
```

You cannot leave a function with a `goto` or `link` statement. All `goto` and `link` statements must target local labels. Only the `return` statement allows you to leave a function before the last statement is reached. The following code gives an error message and the function returns a missing value:

```
  function realsqrt(a) {
     if (a < 0) goto rept;
     k= sqrt(a);
     return(k);
  }
  /* intentional error: label rept not resolved */
  i = realsqrt(-25);
  print "This should be missing value:", i;
rept:
```

The names of all standard functions (except `sizeof()` which is a keyword) can be overridden by user specified functions. For example, if a user specifies a function called `sqrt` a later call to `sqrt` will refer to the user specified function rather than to the standard square root function. This property of CMAT is consistent with C. A call to `sqrt` before the user definition of this function refers to the standard function. Note, however, that simply changing the name of subroutine `rsqrt` to `sqrt` in the example would also affect the call of `sqrt` inside the function body and make that function to an infinitely looping recursive function.

5.13 Runtime Options

5.13.1 Current List of Runtime Options

All keywords used in the `option` statement are case insensitive. The following table lists options that can be used inside the `option` statement:

List of Runtime Options	
Syntax	**Description**
C_FIELDW=*int*	Defines the field width for the output of complex numbers with the `print` statement. Note, this option does not affect the result of the `printf()` and `sprintf()` functions.
CENTER	Each object listed in the `print` statement is printed centered. The NOCENTER option may be used to reset this option to the default. The CENTER option can also be used locally inside the POPTN clause. Note, this option does not affect the result of the `printf()` and `sprintf()` functions.
DEBUG=*string*	Only for (internal) debugging purposes.
DECIMALS=*int*	Defines the number of decimals for the printout of floating point numbers with the `print` statement. Note, this option does not affect the result of the `printf()` and `sprintf()` functions.
ECHO	Echo input in .log output. This is the default. Use option NOECHO for suppressing.
F_FIELDW=*int*	Defines the field width used in the output of floating point numbers with the `print` statement. Note, this option does not affect the result of the `printf()` and `sprintf()` functions.
I_FIELDW=*int*	Defines the field width used in the output of integer numbers with the `print` statement. Note, this option does not affect the result of the `printf()` and `sprintf()` functions.
INDBASE=*int*	(short form: IB) Defines the lower index range. By default, CMAT is compatible with C and therefore uses an indexbase of zero. To be compatible with FORTRAN and many other matrix packages use IB=1 to set the indexbase to one. Only nonnegative integer's are permitted.
SECOND	Run a second version of the algorithm rather than the primary.
LINESIZE=*int*	(short form: LS) Defines the maximum number of characters printed on one line. Default is LS=68.
NALLOC=*int*	set number of largest memory allocations for `lstmem` function. Default is 10.
NAME	By default, scalars printed using the `print` statement are not preceded by the variable name. For scalars the NAME option includes the name of the variable in front of the printed value. When vectors or matrices are printed using the `print` statement the upper left corner of the table (in the intersection of row and column names) is by default either blank or contains an acronym of the type of the matrix (e.g. SYM for symmetric matrix). Specifying the NAME option includes the name of the variable into this corner.
NOCENTER	The NOCENTER option can be used to reset the CENTER option to the default left flushed output.
NODEBUG	Turning off (internal) debugging which was specified with the DEBUG option.
NOECHO	Suppress echo input in .log output. Option ECHO is the default.

List of Runtime Options (Contd.)	
Syntax	**Description**
PRIME	Run the first version of the algorithm rather than the second version. This is the default.
NONAME	This option is needed only to overturn the use of the NAME option back to the default situation.
NOPRINT	The NOPRINT option suppresses all output to the .lst file.
OPT_BS=*int*	Defines the optimal block size used by some LAPACK subroutines. See [15] pp. 50. Default is OPT_BS=64.
PAGESIZE=*int*	(short form: PS) Defines the maximum number of lines printed on one page. Default is PS=60.
PRINT	The PRINT option permits again output to the .lst output file after it was supressed by specifying the NOPRINT option.
RANDCMP	The RANDCMP option specifies that the uniform random of the host compiler is used where CMAT is compiled. The windows version of the Watcom C compiler is the worst choice (based on 16 bit arithmetic).
RANDUNI	The RANDUNI option specifies that the uniform random generator developed by Moore from the RAND Corporation is used. The generator is described at Fishman, 1996, p.605. In SAS Language it is known as RANUNI. This is the default for uniform random numbers.
RANDACM	The RANDACM option specifies that the uniform random generator developed by Schrage in ACM TOMS (1979) is used. This is not a good choice.
RANDKISS	The RANDKISS option specifies that the uniform random generator KISS ("Keep It Simple Stupid") by Marsaglia & Tsang (2002) is used.
RANDLECU	The RANDLECU option specifies that the uniform random generator by L'Ecuyer (1999) is used.
RANDXOR32	The uniform RNG XOR by Marsaglia (2003) is used. Period $2^{32} - 1$
RANDXOR64	The uniform RNG XOR64 by Marsaglia (2003) is used. This is the 64 bit version. Period: $2^{64} - 1$
RANDXORWOW	The uniform RNG XORWOW by Marsaglia (2003) is used. This is probably the best choice available. Period: $2^{192} - 2^{32}$
RANDXOR128	The uniform RNG XOR128 by Marsaglia (2003) is used. Period: $2^{128} - 1$
RANDMWC3	The uniform RNG MWC by Marsaglia (2003) is used. This is a multiply-with-carry algorithm. Period: $2^{128} - 1$
RELZERO=*real*	Defines relative zero criterion: if an entry a_{kl} of a matrix is smaller than the maximum entry in its row or column, it may be set to zero under some circumstances. The value is also used as threshold for the decision when a singular value is considered zero (see below). Default for RELZERO is the machine precision (which is about 10^{-16} for IEEE computers) times 1000.
SEED=*int*	Initializes the *seed* argument of the `rand()` function. Is identical to calling `srand(`*int*`);`.

List of Runtime Options (Contd.)	
Syntax	**Description**
SING=*real*	Defines a criterion used in the singularity test of some linear algebra subroutines. See the corresponding functions for the specific meaning. Default is SING=1e-8.
SPRANGE=*real*	Defines sparsity range which is the threshold of nonzero density in a vector or matrix to switch from dense to sparse storage form. The ratio is defined in terms of significant entries. For example, a diagonal matrix has n significant (diagonal) elements; a lower, or upper triangular and a symmetric matrix have $n*(n+1)/2$ significant elements; otherwise an $m \times n$ matrix has $m*n$ significant elements. If the ratio between the number of nonzeros and significant elements is smaller than SPRANGE, then the matrix is stored sparse, otherwise it is stored in one of the dense forms. Default is SPRANGE=.5
SYMCRIT=*real*	Defines the criterion for the detection of symmetry. If **A** is an $n \times n$ matrix and for all symmetric pairs of elements (a_{ij}, a_{ji}) the absolute value of the difference is larger than the product between SYMCRIT and the sum of the absolute values, then matrix **A** is found symmetric. Default for SYMCRIT is the square root of the machine precision (which is about 10^{-8} for IEEE computers).
USEUTF=*int*	Some functions require the allocation of work space or the use of an utility file. For small need of space the memory allocation is possible and permits faster access, for larger need a utility file may be necessary. This options specifies a number of bytes threshold for the switch from memory allocation to utility file use. Default is USEUTF=1000000.

The formula for the symmetry criterion is,

$$|a_{ij} - a_{ji}| \leq \text{SYMCRIT} * (|a_{ij}| + |a_{ji}|)$$

and that for the relative zero criterion is

$$a_{kl} \leq \text{RELZERO} * \min(\max_{i=1}^{m} a_{il}, \max_{j=1}^{n} a_{kj})$$

The RELZERO criterion is also used in some applications of the singular value decomposition (like the `llls()` function) to decide when a singular value s_j of an $m \times n$ matrix **A** is small enough to be considered zero. Let s_{max} be the largest singular value, then s_j is considered (practically in applications set to) zero when $s_j \leq \text{RELZERO} * s_{max}$.

By default the RANDUNI random generator is used. Note, that the choice of RANDCMP may be the worst since for MS Windows the Watcom `rand()` function only uses a 16 bit RNG. The results of the `diehd()` function suggest that RANDKISS and RANDLECU may be the best choices for uniform random generators.

5.13.2 Print Options

The following table lists options that can be used inside the `poptn` clause affecting only the following output object in the PRINT statement (local influence). Some of these options can also be specified inside the `options` statement and are then valid for all following `print` statements until they are explicitly reset (global influence).

List of Temporary Print Options	
Syntax	**Description**
C_FIELDW=*int*	Defines (locally) the field width for the output of complex numbers.
CENTER	The next object listed in the `print` statement is printed centered.
COLNAME=*name*	Defines column headers for the printed output. Typically, *name* is a vector of strings. If only one name (string) is supplied but the output table has more than one column, a numeric suffix is appended.
DECIMALS=*int*	Defines (locally) the number of decimals for the printout of floating point numbers.
FORMAT="string"	Specifies a format string. The string is either of the form "c" or contains the 'e', 'f', 'g', 'i', 'd', or 's' specifications for each column of the output object. See below for more details of the format string specification.
FORMAT=name	Specifies the output format using a formatting table. See below for more details of the format matrix specification.
F_FIELDW=*int*	Defines (locally) the field width used in the output of floating point numbers.
I_FIELDW=*int*	Defines (locally) the field width used in the output of integer numbers.
LABEL=*name*	Defines row labels for the printed output. Typically, *name* is a vector of strings.
NAME	By default, scalars printed using the `print` statement are not preceded by the variable name. For scalars the NAME option includes the name of the variable in front of the printed value. When vectors or matrices are printed using the `print` statement the upper left corner of the table (in the intersection of row and column names) is by default either blank or contains an acronym of the type of the matrix (e.g. SYM for symmetric matrix). Specifying the NAME option includes the name of the variable into this corner.
NEWPAGE	A page break is performed before printing the next object.
NONAME	This option is needed only to overturn the use of the NAME option back to the default situation.
ROWNAME=*name*	Defines row labels for the printed output. Typically, *name* is a vector of strings. If only one name (string) is supplied but the output table has more than one row, a numeric suffix is appended.
TITLE="string"	Defines the title string used for the output of the next object. More than one title line should be separated by the C style \ `n` line break symbol.
ULINE='char'	Defines the character used for underlining the title. If no title is specified the option is ignored.

List of Temporary Print Options

Specifying the `format` String

By default, all table elements are scaled with respect to the maximum absolute entry in the whole matrix which is member of the attribute structure of the matrix. For example, the decision for 'e' or 'f' output format is based on this maximum absolute entry. This format is then used for all numbers except for the following special case: If format 'f' was selected with a specific precision and a matrix entry must be printed which is too 'small' with respect to the maximum absolute entry used for the scaling, that small number may be printed in 'e' format. If no `format` option is specified all columns of the table have the same size.

The special form `format="c"` means that each column of the output matrix is scaled with respect to the maximum absolute entry in this column. For floating point numbers some of the columns may be printed in 'e' and other in 'f' format.

If the format string is not of the form "c" it must contain the 'e', 'f', 'g', 'i', 'd', or 's' specifications for **each** column of the output table. If the column contains only strings, the 's' format may be specified. The character 's' may be followed by an int number specifying the field width used for formatting the string. If the column contains only integer data the 'i' and (the alias) 'd' format can be used. The format type may be followed by an int number specifying the field width used for formatting the number. If the column contains floating point numbers (real or complex) the 'e', 'f', or 'g' format must be used. For complex numbers two specifications are needed, the first for the real, the second for the imaginary part. Each of the format types may be followed by two int numbers separated by a period '.', the first specifying the field width and the second specifying the precision used in printing the floating point number.

For example, the string may be of the form "e16.6", "f12.4", or "g16.4" for floating point numbers and like "d2" for integers. Note that the C style % sign is dropped and the character specification preceeds the numeric. The typical C style format specification may be used with the `printf` and `sprintf` functions. You may *cast* the data type of the output object if it is not obvious in a more difficult CMAT application.

Specifying the `format` Matrix

The `format` option can also be used to specify a simple variable name (no expressions can be used) defining a formatting matrix. Each row of the format matrix specifies the format properties of one column of the output table:

1. The (integer) column number of the output table for which the remaining information of the row is valid.
2. The (integer) width of the column of the table, not including two separating spaces. Specifying a missing value or zero means that the minimum column size is used.
3. The (string) format type ('e', 'f', 'g', 'i', 'd', or 's') of the column.
4. The (integer) field width of the number to be printed.
5. Only for floating point numbers: the (integer) precision of the number.
6. For complex numbers there may be three more columns attached to the format matrix specifying the format type, field width, and precision of the imaginary part of the complex numbers in that column.

Columns of the output table for which no format information is given in the format matrix are formatted by default based on the maximum entry in that column.

For example, the following formatting matrix specifies the format used for the output of the first three columns of the 4×4 matrix of integers, the fourth column is printed using the default format:

```
a = [ 15 ,                          L |          1         2          3         4
      22  51];                     ---------------------------------------------
aa = [ a,                           1 |   1.5e+001
       a a];                        2 |   2.2e+001     51.00
form = [ 1   8  "e"   8  1,         3 |   1.5e+001      0.00        15.0
         2   8  "f"   8  2,         4 |   2.2e+001     51.00        22.0        51
         3  10  "f"  10  1 ];
print poptn format=form : aa;
```

Example: A Simple Integer Matrix

The following example illustrates some easy options for controlling the output of an integer matrix. First we show the input and default output of the 2×2 lower triangular matrix **A**,

```
a = [ 15 ,                    L |    1    2
      22  51];               ------------
print a;                      1 |   15
                             2 |   22   51
```

The following statement changes the field width of all columns

```
print poptn i_fieldw=6 : a;
```

resulting in the left flushed output

```
 L |       1       2
--------------------
 1 |      15
 2 |      22      51
```

and the center option centers the output with respect of the current linesize specification,

```
print poptn center i_fieldw=6 : a;
```

```
                      LOW |       1       2
                     ----------------------
                        1 |      15
                        2 |      22      51
```

We also want to be sure that the `poptn` terminating colon may be used inside a title string specification

```
/* colon inside title string should be permitted */
print poptn center title="Smokers: Orders" uline='-' : a;
```

We also choose to underline the title string with dashes,

```
                         Smokers: Orders
                         ---------------

                    LOW |         0         1
                   ---------------------------
                      0 |        15
                      1 |        22        51
```

Now we illustrate how row and column names and row labels may be specified,

```
rnam= [ "Smoker" "Nonsmoker" ];
cnam= [ "Price"  "Volume" ];
labl= [ "Protestantic Smokers", "Catholic Nonsmokers" ];
print poptn rowname=rnam colname= cnam label=labl : a;
```

resulting in the following output

```
        LOW |    Price   Volume
------------------------------------------------------
     Smoker |       15           Protestantic Smokers
  Nonsmoker |       22       51  Catholic Nonsmokers
```

Example: Formatting a Real Matrix

The following matrix contains differently scaled columns:

```
aa = [ .001  5001.53  1115566.3,
       .005    85.42    -7891.5 ];
option name;
print aa;
```

If `name` is specified inside an `option` statement it is valid for all following `print` statements until `noname` is specified in an `option` statement. Therefore, the upper left corner now contains the name `aa` of the matrix:

```
 aa |           1           2           3
----------------------------------------
  1 |   1.000e-03      5001.5   1115566.3
  2 |   5.000e-03        85.4     -7891.5
```

If no `format` option is specified all columns of the table have the same width and all entries are scaled with respect to the largest. The same format type ('e' or 'f') is used with the exception that small numbers may be printed in 'e'. The following statement specifies that separate column formatting should be applied to the output of `aa`:

```
print poptn format="c" : aa;
```

```
 aa |         1         2           3
-------------------------------------
  1 |   0.00100   5001.53   1115566.3
  2 |   0.00500     85.42     -7891.5
```

Now we just specify the formatting type of the three columns to be 'e', 'f', and 'e':

```
print poptn format="e f e" : aa;
```

```
 aa |            1           2            3
-----------------------------------------------
  1 |   1.0000e-03   5001.5300   1.1156e+06
  2 |   5.0000e-03     85.4200  -7.8915e+03
```

The following statement not only specifies the format type 'e', 'f', and 'e', but also field width and precision for column 1 and three,

```
print poptn format="e8.1 f f10.1" : aa;
```

```
 aa |         1           2           3
-----------------------------------------
  1 |   1.0e-03   5001.5300   1115566.3
  2 |   5.0e-03     85.4200     -7891.5
```

The following example uses a format matrix

```
form = [ 1   8  "e"   8  1,
         2   8  "f"   8  2,
         3  10  "f"  10  1 ];
print poptn format=form : aa;
option noname;
```

and results in the following output

```
 aa |         1         2           3
------------------------------------------
  1 |   1.0e-03   5001.53   1115566.3
  2 |   5.0e-03     85.42     -7891.5
```

Formatting a Complex Matrix

The following statements generate two 9 vectors `y1` and `y2` containing the values of the inverse trigonometric tangent (arctan) and of the inverse hyperbolic tangent (arctanh) function of arguments $x = -2, -1.5, -1., \ldots, 2.$ After concatenating the three vectors into a 9×3 matrix

```
x = [ -2. : .5 : 2. ]‘;
y1 = atan(x); y2 = atanh(x);
xyy = x -> y1 -> y2;
print xyy;
```

the default print output yields the following table which is split into two panels due to the default linesize specification (LS=68):

```
   |                      1                       2
-----------------------------------------------------
 1 |   -2.000e+00 + 0.00000i   -1.107e+00 + 0.00000i
 2 |   -1.500e+00 + 0.00000i   -9.828e-01 + 0.00000i
 3 |   -1.000e+00 + 0.00000i   -7.854e-01 + 0.00000i
 4 |   -5.000e-01 + 0.00000i   -4.636e-01 + 0.00000i
 5 |    0.000e+00 + 0.00000i    0.000e+00 + 0.00000i
 6 |    5.000e-01 + 0.00000i    4.636e-01 + 0.00000i
 7 |    1.000e+00 + 0.00000i    7.854e-01 + 0.00000i
 8 |    1.500e+00 + 0.00000i    9.828e-01 + 0.00000i
 9 |    2.000e+00 + 0.00000i    1.107e+00 + 0.00000i

   |                      3
---------------------------
 1 |   -5.493e-01 + 1.57080i
 2 |   -8.047e-01 + 1.57080i
 3 |  -1.798e+308 + 0.00000i
 4 |   -5.493e-01 + 0.00000i
 5 |    0.000e+00 + 0.00000i
 6 |    5.493e-01 + 0.00000i
 7 |   1.798e+308 + 0.00000i
 8 |    8.047e-01 + 1.57080i
 9 |    5.493e-01 + 1.57080i
```

Since the values of atanh(1) and atanh(-1) are infinite the 'e' format is used by default. Also, since $atanh(x)$ is complex for $|x| > 1$ the data type of the whole table is complex, even though most of the numbers are real.

The specification `format="c"` can be used for a default columnwise printout:

```
print poptn format="c" : xyy;
```

resulting in a table where the first two columns of the table are printed in 'f' format but the last is still printed in 'e' format:

```
   |                    1                    2
-----------------------------------------------
 1 |  -2.00000 + 0.00000i  -1.10715 + 0.00000i
 2 |  -1.50000 + 0.00000i  -0.98279 + 0.00000i
 3 |  -1.00000 + 0.00000i  -0.78540 + 0.00000i
 4 |  -0.50000 + 0.00000i  -0.46365 + 0.00000i
 5 |   0.00000 + 0.00000i   0.00000 + 0.00000i
 6 |   0.50000 + 0.00000i   0.46365 + 0.00000i
 7 |   1.00000 + 0.00000i   0.78540 + 0.00000i
 8 |   1.50000 + 0.00000i   0.98279 + 0.00000i
 9 |   2.00000 + 0.00000i   1.10715 + 0.00000i

   |                    3
------------------------
 1 |  -0.54931 + 1.57080i
 2 |  -0.80472 + 1.57080i
 3 |    .      + 0.00000i
 4 |  -0.54931 + 0.00000i
 5 |   0.00000 + 0.00000i
 6 |   0.54931 + 0.00000i
 7 |    .      + 0.00000i
 8 |   0.80472 - 1.57080i
 9 |   0.54931 - 1.57080i
```

To fit the whole table into one panel without changing the default linesize we may also specify the restricted field width and precision for each column:

```
print poptn format="f6.2 f4.1 f6.2 f4.1 e10.2 f5.2" : xyy;
```

Note, that for complex numbers two specifications were needed (the first for the real and the second for the imaginary part).

```
   |              1              2                   3
-------------------------------------------------------
 1 |   -2.00 + 0.0i   -1.11 + 0.0i  -5.49e-001 + 1.57i
 2 |   -1.50 + 0.0i   -0.98 + 0.0i  -8.05e-001 + 1.57i
 3 |   -1.00 + 0.0i   -0.79 + 0.0i      .      + 0.00i
 4 |   -0.50 + 0.0i   -0.46 + 0.0i  -5.49e-001 + 0.00i
 5 |    0.00 + 0.0i    0.00 + 0.0i   0.00e+000 + 0.00i
 6 |    0.50 + 0.0i    0.46 + 0.0i   5.49e-001 + 0.00i
 7 |    1.00 + 0.0i    0.79 + 0.0i      .      + 0.00i
 8 |    1.50 + 0.0i    0.98 + 0.0i   8.05e-001 - 1.57i
 9 |    2.00 + 0.0i    1.11 + 0.0i   5.49e-001 - 1.57i
```

We may also print some column headers:

```
cnam = [ "x" "atan" "atanh" ];
print poptn colname= cnam
            format="f6.2 f4.1 f6.2 f4.1 e10.2 f5.2" : xyy;
```

```
   |             x        atan(x)            atanh(x)
-------------------------------------------------------
 1 |   -2.00 + 0.0i   -1.11 + 0.0i  -5.49e-001 + 1.57i
 2 |   -1.50 + 0.0i   -0.98 + 0.0i  -8.05e-001 + 1.57i
 3 |   -1.00 + 0.0i   -0.79 + 0.0i      .      + 0.00i
 4 |   -0.50 + 0.0i   -0.46 + 0.0i  -5.49e-001 + 0.00i
 5 |    0.00 + 0.0i    0.00 + 0.0i   0.00e+000 + 0.00i
 6 |    0.50 + 0.0i    0.46 + 0.0i   5.49e-001 + 0.00i
 7 |    1.00 + 0.0i    0.79 + 0.0i      .      + 0.00i
 8 |    1.50 + 0.0i    0.98 + 0.0i   8.05e-001 - 1.57i
 9 |    2.00 + 0.0i    1.11 + 0.0i   5.49e-001 - 1.57i
```

and finally we print a centered table with a title and the variable name in its corner

```
print poptn center name colname= cnam
            title="Table of atan and atanh values in [-2,2]"
            format="f6.2 f4.1 f6.2 f4.1 e10.2 f5.2" : xyy;
```

```
        Table of atan and atanh values in [-2,2]

 xyy |             x        atan(x)            atanh(x)
-------------------------------------------------------
   1 |   -2.00 + 0.0i   -1.11 + 0.0i  -5.49e-001 + 1.57i
   2 |   -1.50 + 0.0i   -0.98 + 0.0i  -8.05e-001 + 1.57i
   3 |   -1.00 + 0.0i   -0.79 + 0.0i      .      + 0.00i
   4 |   -0.50 + 0.0i   -0.46 + 0.0i  -5.49e-001 + 0.00i
   5 |    0.00 + 0.0i    0.00 + 0.0i   0.00e+000 + 0.00i
   6 |    0.50 + 0.0i    0.46 + 0.0i   5.49e-001 + 0.00i
   7 |    1.00 + 0.0i    0.79 + 0.0i      .      + 0.00i
   8 |    1.50 + 0.0i    0.98 + 0.0i   8.05e-001 - 1.57i
   9 |    2.00 + 0.0i    1.11 + 0.0i   5.49e-001 - 1.57i
```

5.14 Debugging Tools

The following functions can be used for debugging CMAT code:

- `r = attrib(z<,"`*prop*`">)`: If there is no second argument specified, the call of this function returns and prints a full table of attributes of the scalar, vector, or matrix z. If the second argument is specified as "nopr" the output is suppressed and the table is returned as a $n \times 2$ matrix object. Note, that the `attrib()` function only prints or returns the prior set values of the attribute structure and does not compute any of the values during its call. For example, executing `x = a \ b;` for an $n \times n$ matrix $\mathbf{A}$ usually fills the condition and determinant fields of the attribute structure automatically.
- `lstmem(i)`: Prints a detail and summary table of the memory used by objects (scalars, vectors, matrices, tensors) which were defined in former CMAT commands. If the second (optional) argument i is not specified or chosen to zero, only the summary table is printed. For $i \neq 0$ additionally the detail table is printed.
- `lststk(i)`: Prints a summary table of the stack content generated in former CMAT commands. If the second (optional) argument i is not specified or chosen to zero, the whole stack is printed, otherwise only the last i stack entries are printed.
- `lstvar(i)`: Prints a detail and summary table of the variable tables defined in former CMAT commands. If the second (optional) argument i is not specified or chosen to be zero, only the summary table is printed. For $i \neq 0$ additionally the detail table is printed.

5.15 Concept of Options Vector and Options Matrix

Many of the functions described in the reference manual permit an options input vector which can be used to specify additional options by the user. Each index location of such a vector corresponds to a specific option. Since CMAT permits mixed data type objects, there may be entries of the options vector which are of string type whereas other entries are of numeric type. The user should read the document carefully before setting an option. In most cases each option has its default value which should apply to most of the situations the function is applied. It is, however, appropriate to give the user an opportunity to select inbetween other choices and he/she may use the options vector to override the default setting(s). Specifying a missing value (in CMAT a period) for an entry of the options vector means the default setting should be used. For many applications a 2-step approach for setting the options vector seems to be appropriate:

1. Create a vector of sufficient length containing only missing values using the `optn = cons(n,1,.);` statement which creates a column vector containing n missing values.
2. Override missing values of those vector entries for which other than default options should be used by the specific specifications.

A large number of functions in CMAT expect the specification of a two column options matrix. This is a matrix of mixed data type (string and numerical) where the first column contains a string (in quotes) specifying the option and the second column is either not specified (zero) or contains a options value which can be either a string or a numeric value. The following is an example for a two column options matrix used in the call of the `glmixd` function:

```
optn = [ "print"         4,    /* lots of printed output */
         "link"    "logit",    /* link function */
         "efdis"  "normal",    /* distribution of random effects */
         "yscale"      "n",    /* scale of response is nominal */
         "idvar"         2,    /* column number for ID variable */
         "ctvar"         6,    /* column number for count variable */
         "cl"       "wald",    /* Wald confidence intervals */
         "nquad"        10,    /* number of quadrature points */
         "tech"   "trureg" ];  /* optimization technique */
model = "3 = 6 7 8 9";
class = [ 2 3 ];
< gof,par,ase,ci,cov> = glmixd(tvsfp,model,optn,class);
```

5.16 Concept of the Model String Argument

For many statistical functions, the analysis model can specified in form of a string containing column numbers for variables. The model string specifies which variables (columns) are independend (covariates) and which are dependend (response) and can be of the form

- for functions that perform exploratory (without response) modeling (unsupervised learning):

$$X_{effect_1} \ldots X_{effect_n}$$

- for functions that perform univariate or multivariate predictive modeling (supervised learning):

$$Y_1 \ldots Y_r = X_{effect_1} \ldots X_{effect_n}$$

  For multivariate predictive modeling, all r response variables $Y_1, \ldots, Y_r$ are to be predicted simultaneously by the n predictor effects. Some functions like `glmod`, which are designed for univariate predictive modeling permit the specification of multiple response variables $Y_1, \ldots, Y_r$. In that case, the parameters of the effects are estimated w.r.t. each response variable Y_i in r analyses separately.

Here X_{effect_i} may be one of the following:

- a single variable x_i is an effect
- effects that are interactions among multiple x variables, separated by the $*$ operator, $x_i * \ldots * x_l$
- nested effects: that are single variables or interactions followed by a list of variables inside parentheses, $(x_i \ldots x_l)$. Note that only categorical variables, i.e. variables listed in the CLASS argument, without interactions can be listed inside parentheses.

There are a number of additional features that can make it easier to use such kind of model specifications:

- You may use the | (bar) operator, for the shorter notation of `model = "y = a b a*b"` by `model="y = a | b"`. Note, that using more than two colon operators in one term may create a large number of effects, e.g. `"a|b|c|d"` creates the following 15 effects: `"a b a*b c a*c b*c a*b*c d a*d b*d c*d a*b*d a*c*d b*c*d a*b*c*d"` For limiting the maximum number of variables in the effects, the @ operator with an integer can be used, e.g. `"a|b|c|d@2"` creates only 10 effects: `"a b a*b c a*c b*c d a*d b*d c*d"`.
- You may use the colon operator to express a successive number single variable effects, e.g. `"y= 1 : 3 * 5"` is the same as `"y= 1 2 3*5"`.
- If a string has to be stretched over more than one line of code, the line break must be indicated with a \ symbol at the end of each line.
- Instead of specifying the entire model in one long string variable you may use a vector of strings. However, each effect must be entirely contained in one vector entry and cannot be continued with the next entry. E.g. the above example could be split into the vector `v` as `v[1]="y="; v[2]="1 2"; v[3]="3*5";` . Note, that the order of the effect listing is important for stepwise regression and Type I estimates.

The syntax of the `model` string argument is the same for a large number of statistical functions, e.g. the `glmod`, `glim`, `glmixd`, and `impute` functions.

5.17 Concept of the Class Argument

For some of the statistical functions the data matrix which is subject to analysis may contain categorical variables. Categorical variables have a small number of values (levels) and are coded either as integers (starting from zero or one for each level) or strings. The scale of a categorical variable (levels) can be either nominal or ordinal. The analysis methods usually internally replace each nominal or ordinal variable by a number of binary variables which have only two values, zero and one.

When in CMAT the model argument contains categorical variables, i.e. variables which have nominal or ordinal scale, they should be listed with their column numbers in a `class` argument. The `class` argument contains either a single integer scalar or a vector of integer referring to column (variable) numbers in the data matrix.

Currently, all variables listed in the `class` argument are treated as nominal. Nominal variables with K levels are coded internally either in form of K or $K-1$ binary variables using one of the two coding designs:

1. *rankdeficient design*: A nominal variable with K levels is replaced by K binary variables. For example, for $K = 4$ the following coding schedule is applied:

Level of Variable	**Coding of Binary Variables**			
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

   The *rankdeficient design* results in design matrices $\mathbf{X}$ where the K columns have rank $K-1$, i.e. $\mathbf{X}$ is not of full rank K and singular. The advantage of the rank deficient design is good interpretability of the results. The disadvantage of it are problems with inverting singular matrices such like $\mathbf{X}^T\mathbf{X}$. Only a few functions like `glmod` and `glim` in CMAT are able to deal with the rankdeficient design.

2. *full-rank design*: A nominal variable with K levels is replaced by $K-1$ binary variables. For example, for $K = 4$ the following coding schedule is applied:

Level of Variable	**Coding of Binary Variables**		
1	1	0	0
2	0	1	0
3	0	0	1
4	0	0	0

   The *full-rank design* results in design matrices $\mathbf{X}$ where the $K-1$ columns have full column rank, i.e. with some further usually mild assumptions like independence of rows and $N > n$ the $K-1$ columns of the matrix have full rank and $\mathbf{X}$ may not be not singular. The disadvantage of the full-rank design is limited interpretability of the results. The advantage of it are easier numerical treatment of nonsingular matrices such like $\mathbf{X}^T\mathbf{X}$. Many more functions in CMAT are able to deal with the full-rank design than are with the rankdeficient design.

For the future of CMAT we also plan to analyze ordinal variables. An ordinal variable with K levels is replaced by either K or $K-1$ binary variables. For example, for $K = 4$ the following rank-deficient coding schedule is applied:

Level of Variable	**Coding of Binary Variables**			
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1

For example, for $K = 4$ the following full-rank coding schedule is applied:

Level of Variable	**Coding of Binary Variables**		
1	0	0	0
2	1	0	0
3	1	1	0
4	1	1	1

The reason for treating categorical variables as nominal in the past was, that the correct treatment of ordinal variables usually requires constrained (nonnegative) optimization methods for the parameter estimation.

5.18 Saving Computer Resources

5.18.1 Avoid Scalar Operations

For a finite difference discretization of the integral operator

$$f(s) = \int K(s,t)x(t)dt$$

with 2-dimensional kernel $K(s,t)dt$,

$$K(s,t) = \begin{cases} s(t-1) & \text{if } 0 \le s \le t < 1 \\ t(s-1) & \text{if } 0 \le t < s \le 1 \end{cases}$$

we compute an $m \times n$ matrix $\mathbf{A}$

$$A(i,j) = \begin{cases} k*(s_i)*(t_j-1) & \text{if } i \le j \\ k*(t_j)*(s_i-1) & \text{if } i > j \end{cases}$$

where $s_i = \frac{i}{m+1}$, $t_j = \frac{j}{n+1}$, and $k = \frac{1}{n+1}$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. For $m = 200$ and $n = 100$ we use two vectors `sr` and `tc`:

```
m = 200; n = 100;
mp = m + 1; np = n + 1;
hm = 1. / mp; hn = 1. / np;
sr0 = hm * [ 1 : m ]; sr2 = sr[1:n];
tc0 = hn * [ 1 : n ]; /* print sr0,tc0; */
sr1 = sr0 - 1.; tc1 = tc0 - 1.;
```

The first example illustrates the creation of the $m \times n$ matrix with all scalar operations:

```
tim1 = time("clock");                                   Time 1: 217.360
for (i = 1; i <= m; i++)
for (j = 1; j <= n; j++)
a[i,j] = (i <= j) ? sr0[i] * tc1[j]
                  : sr1[i] * tc0[j];
rec1 = hn * a;
tim1 = time("clock") - tim1;
print "Time 1:", tim1;
```

The second example uses two Kronecker products for obtaining $m \times n$ matrix $\mathbf{A}$ and $n \times n$ matrix $\mathbf{B}$ and then resets the upper triangle of matrix $\mathbf{A}$ using scalar operations:

```
tim2 = time("clock");                                    Time 2: 104.969
a = sr1‘ @ tc0;
b = sr2‘ @ tc1;
for (i = 1; i <= n; i++)
for (j = i; j <= n; j++) a[i,j] = b[i,j];
rec2 = hn * a;
tim2 = time("clock") - tim2;
print "Time 2:", tim2;
```

The third example creates matrices **A** and **B** as in example 2 but then resets the upper triangle of matrix **A** using operations with index vectors:

```
tim3 = time("clock");                                    Time 3: 4.7340
a = sr1‘ @ tc0;
b = sr2‘ @ tc1;
for (i = 1; i < n; i++) {
  ir = [i : n];
  a[i,ir] = b[i,ir];
}
rec3 = hn * a;
tim3 = time("clock") - tim3;
print "Time 3:", tim3;
```

The example illustrates how vector operations can reduce the computer time significantly. Increasing the number of rows to $m = 300$ shows the following result:

```
Time 1: 486.813
Time 2: 162.953
Time 3: 6.9850
```

5.18.2 Avoid Symmetry Swapping

When creating large matrices elementwise should be avoided. Instead matrix or vector operations should be used whenever possible to save expensive `for` loops by the interpreter. Creating matrices elementwise can be especially slow when the symmetry property of a matrix is changed with each assignment. This may be avoided by creating only the lower triangle of the matrix and then using the `(tri2sym)` cast for setting the upper triangle symmetrically. The following example illustrates the difference in the approach. The following code shows four different versions for the elementwise computation of a symmetric $n \times n$ matrix with a Gaussian radial basis Kernel function

$$K(x_i, x_j) = y_i y_j(\exp(-\frac{(x_i - x_j)^T(x_i - x_j)}{n_{var}}) + 1.)$$

as it may be needed for an `svm` application. Ofcourse we could obtain the same result much faster by simply using the `svmat()` function.

First we read the `heart_scale` data set and decide which column is the response and which are the predictor variables.

```
data = rspfile("heart_scale");
```

```
/* n = nrow(data); */ n = 100;
y = data[,1]; x = data[,2:14];
fac = 1. / 13.;
```

1. Initialize with zero and set symmetric pair, i.e. swapping inbetween symmetric and unsymmetric matrices inside the inner loop:

```
/* (1) sparse: symmetric swap */
tim = time("clock");
q = cons(n,n,0.);
for (i = 1; i <= n; i++) {
  /* print "row=", i; */
  u = x[i,];
  for (j = 1; j <= i; j++) {
    w = u - x[j,]; t = ssq(w);
    r = y[i] * (exp(-fac * t) + 1.) * y[j];
    q[i,j] = q[j,i] = r;
} }
tim1 = time("clock") - tim;
print tim1;
```

2. Initialize with one and set symmetric pair, i.e. swapping inbetween symmetric and unsymmetric matrices inside the inner loop:

```
/* (2) sparse: unsymmetric */
tim = time("clock");
q = cons(n,n,1.);
for (i = 1; i <= n; i++) {
  /* print "row=", i; */
  u = x[i,];
  for (j = 1; j <= i; j++) {
    w = u - x[j,]; t = ssq(w);
    r = y[i] * (exp(-fac * t) + 1.) * y[j];
    q[i,j] = q[j,i] = r;
} }
tim2 = time("clock") - tim;
print tim2;
```

3. Initialize with zero and set only lower triangular entry inside the inner loop:

```
/* (3) dense: symmetric swap */
tim = time("clock");
q = cons(n,n,0.);
for (i = 1; i <= n; i++) {
  /* print "row=", i; */
  u = x[i,];
  for (j = 1; j <= i; j++) {
    w = u - x[j,]; t = ssq(w);
    r = y[i] * (exp(-fac * t) + 1.) * y[j];
    q[i,j] = r;
```

```
} }
q = (tri2sym)q;
tim3 = time("clock") - tim;
print tim3;
```

4. Initialize with one and set only lower triangular entry inside the inner loop:

```
/* (4) dense: unsymmetric */
tim = time("clock");
q = cons(n,n,1.);
for (i = 1; i <= n; i++) {
  /* print "row=", i; */
  u = x[i,];
  for (j = 1; j <= i; j++) {
    w = u - x[j,]; t = ssq(w);
    r = y[i] * (exp(-fac * t) + 1.) * y[j];
    q[i,j] = r;
} }
q = (tri2sym)q;
tim4 = time("clock") - tim;
print tim4;
```

For $n = 50$ (a dense matrix with 2500 nonzero entries) the four approaches take t1=18.031, t2=19.954, t3=3.1870, t4=4.4220 seconds, and for $n = 100$ (a dense matrix with 10000 nonzero entries) the four approaches take t1=280.187, t2=296.454, t3=32.422, t4=51.110 seconds in my 200 MHz Pentium processor.

The following code shows how to obtain the same result using the `svmat()` function.

```
data = rspfile("heart_scale");
n = nrow(data);
modl = "1 = 2 : 14";
class = 1;
optn = cons(20,1,.);
optn[ 1] = 3;         /* ipri */
optn[ 4] = 4;         /* kfun=RBF2 */
optn[ 8] = 1. / 13.;  /* pkf1 */
optn[10] = 1;         /* bcop */
hes = svmat(data,modl,optn,class);
```

5.18.3 Saving Core Memory by Dumping Objects into Files

Large objects (i.e. vectors or matrices) may be saved in files using the `obj2fil()` function and later may be moved back into core using the `fil2obj()` function. The `libname` specification enables the specification of lookup directories, that means, the filenames (without extension) can be used just like variable names in CMAT statements.

5.18.4 Use Larger Objects

For some test, we want to create a large sparse matrix:

```
nr = nc = 5;
rind = [ 1  2  3   5      7     ];
cind = [ 1  2  3   1  4   2  5  ];
vals = [ 5. 5. 5.  1. 5.  2. 5. ];
c = spmat(nr,nc,rind,cind,vals,"rowsy");
print "Matrix C=",c;
```

Concatenating two thousand 5×5 matrices takes a very long time:

```
t1 = time("clo");
sigma = c;
for (i = 2; i <= 2000; i++) sigma = sigma \> c;
dt1 = time("clo") - t1;
```

Instead we first concatenate onehundred 5×5 matrices, then ten 500×500 matrices, and after that just two 5000×5000 matrices:

```
t1 = time("clo");
/* from 5 to 500 */
sigma = c;
for (i = 2; i <= 100; i++) sigma = sigma \> c;
/* from 500 to 5000 */
c = sigma;
for (i = 2; i <= 10; i++) sigma = sigma \> c;
/* from 5000 to 10000 */
sigma = sigma \> sigma;
dt1 = time("clo") - t1;
```

The speedup is very impressive.

Chapter 6

Summary of Operators, Keywords, and Functions

6.1 General Remarks

The list of functions in CMAT contains many of the standard functions of a C compiler library (see [354], chapter 11). However, many of these functions were extended to both, complex arithmetic and matrix and tensor notation. In a few cases the functions had to be modified for practical purposes. For example, many functions have a minimum and a maximum number of arguments, and arguments inbetween those two ranges may be set to missing values permitting internal default settings of those arguments.

Some of the traditional one-argument functions (like `abs`) are trivially extended to matrix notation, so that the scalar function is applied to every element of the object and the result is an object of the same dimension. Some of the traditional 2-argument functions (like `atan2` or `fmod`) are extended in the following most trivial way:

- If one of the two arguments is scalar and the other argument is a matrix or vector, then the function is performed using the scalar and each entry of the other operand. The result is of the same size as the matrix or vector argument. The function is performed only on numeric data, string data are not processed.
- If both arguments are vectors or matrices, then both must have the same dimension (same number of rows and columns). The function is performed pairwise with corresponding pairs of entries of the two operands.

When the traditional 2-argument functions `max` and `min` are being called with only one vector, matrix, or tensor argument, it will return the entry of the data object which has the largest resp. smallest value.

Some of the functions can deal with missing values (coded as a period) in an intelligent way. Usually this is mentioned in the `Restrictions` part of each function in the reference manual. Usually, a missing value is returned if one of the non-default arguments is a scalar with missing value. Note, that for a number of functions the run time option SECOND will execute an alternative implementation not necessarily one related to the LAPACK software product and not necessarily only for linear algebra functions. If you choose for some reason to select the SECOND option for a specific function call, you are adviced to turn off this option afterward by specifying the PRIME option since there is still some not fully tested code among the alternatives.

Some of the larger functions permit the input of options vectors or two-column options matrices as input arguments. Each entry of the options vector corresponds to a specific option and must be set to a missing value

to permit the default value to be used. The default value must not necessarily be zero. Therefore it is highly recommended to initialize an options vector with missing values like `optvec = const(10,1,.);` to avoid that unexpected zero settings are being used.

6.2 Tables of Operators and Functions

6.2.1 Keywords

Keywords	
Keyword	**Short Description**
`break`	terminates loop or case in switch statement
`case`	case label in `switch` statement
`char`	cast to char data type
`complex`	cast to complex data type
`conj`	cast to conjugate complex
`continue`	continues with next loop in `for` for `while`
`default`	default label in switch statement
`else`	introduces the alternative to `if`
`exit`	terminates excution of CMAT
`for`	introduces loop statement
`free`	free memory belonging to a variable
`function`	introduces function definition
`global`	defines global arguments in function definition
`gnuplot`	start of *gnuplot* script
`goto`	executes jump to a label
`gpend`	end of *gnuplot* script
`herm`	cast matrix to be Hermitian
`if`	introduces conditional statement
`imag`	cast for imaginary part of complex number
`int`	cast for integer data type
`link`	specifying label as identifier for `return` jump
`list`	defining data lists (array of objects)
`option`	used for specifying runtime options
`poptn`	used for specifying runtime print options
`print`	generic `print` statement
`psd`	cast matrix to be positive semi-definite
`rename`	renaming variable or user function
`real`	cast data type to floating point real
`return`	returns either to function call or link statement
`sizeof`	returns size of object (`sizeof` is no function)
`struct`	defining data structs
`switch`	for conditional jump to a set of case labels
`symm`	cast matrix to be symmetric
`tri2sym`	cast lower or upper triangular matrix to symmetric
`void`	may be used for type of function definition
`while`	introduces loop statement

6.2.2 Operators

Operators	
Op	**Meaning**
/* . */	comment
( )	grouping terms in expression
	function parameters, arguments
[]	matrix literal
[""]	modified matrix literal
<>	function returns
{ }	compound statement
Unary Arithmetic Operators	
+	positive sign
−	negative sign
!	logical negation
~	one's complement
++	increment
−−	decrement
'	transpose
.'	nonconjugate transpose
Binary Arithmetic Operators	
+	addition
−	subtraction
*	multiplication
/	forward division
\	backward division
%	remainder int division
**	power
@	Kronecker product
.*	elementwise multiplication
./	elementwise forward division
.\	elementwise backward division
.**	elementwise power
Shift, Relational, and Logical	
<<	left shift
>>	right shift
&	bitwise-and operation
|	bitwise-inclusive-or operation
^	bitwise-exclusive-or operation
&&	logical-and operation
||	logical-or operation
<	less-than relation
>	greater-than relation
<=	less-than-or-equal relation
>=	greater-than-or-equal relation
==	equal-to relation
!=	not-equal-to relation
.<	elementwise less-than relation
.>	elementwise greater-than relation
.<=	elementwise less-than-or-equal-to rel.
.>=	elementwise greater-than-or-equal-to rel.
.==	elementwise equal-to relation
.!=	elementwise not-equal-to relation
? :	Conditional Operator

Operators	
Op	**Meaning**
Concatenation Operators	
−>	horizontal from left to right
<−	horizontal from right to left
|>	vertical from top to bottom
<|	vertical from bottom to top
\>	diagonal from left top to right bottom
</	diagonal from left bottom to right top
Assignment Operators	
+=	addition assignment
−=	subtraction assignment
*=	multiplication assignment
/=	forward division assignment
\=	backward division assignment
%=	remainder assignment
@=	Kronecker product assignment
<<=	left-shift assignment
>>=	right-shift assignment
&=	bitwise-and assignment
|=	bitwise-or assignment
^=	bitwise-exclusive-or assignment
Subscript Reduction Operators	
+	sum
*	product
|+|	sum of absolute values
|*|	product of absolute values
**	sum-of-squares
<>	maximum value
><	minimum value
<:>	index of maximum value
>:<	index of minimum value
<|>	maximum absolute value
>|<	minimum absolute value
<!>	index of max. abs. value
>!<	index of min. abs. value
Sorting and Ranking	
<	ascending values
>	descending values
<:	ranks of ascending values
>:	ranks of descending values
<|	ascending absolute values
>|	descending absolute values
<!	ranks of ascending absolute values
>!	ranks of descending absolute values
~	permute randomly

6.2.3 Constants and Information

Constants and Information		
Function	**Spec**	**Short Description**
`attrib(`z `<,"`*prop*`" >)`	prop=	returns attributes of object
	"nopr"	do not print table of object attributes
	"name"	name of object (first argument)
	"otyp"	object type
	"dtyp"	data type
	"styp"	storage form
	"nrow"	number of rows (same as nrow(z) function)
	"ncol"	number of cols (same as ncol(z) function)
	"lbw"	lower band width
	"ubw"	upper band width
	"slen"	(maximum) length of strings
	"nstr"	number of strings
	"nmis"	number of missing values in object
	"nzer"	number of nonzero values in object
	"vmin"	smallest value
	"vmax"	largest value
	"nrm2"	2-norm (Frobenius or Euclidean) of object
	"rcond"	reciprocal condition of matrix
	"det"	determinant of matrix
`all(`a`)`		check for all entries nonzero
`any(`a`)`		check for any nonzero elements
b `= bmi(`$w\_kg, h\_met <, bmi <, optn >>$`)`		*body-mass-index*
b `= branks(`a`)`		return tied and bivariate ranks of `a`
b `= chngtxt(`$a, old, new <, numb >$`)`		change text in string data to other text
$clab$ `= clabel(`a`)`		return column labels `clab` of `a`
b `= clabel(`$a, clab$`)`		assign labels `clab` to columns of `a`
$cnam$ `= cname(`a`)`		return column names `cnam` of `a`
b `= cname(`$a, cnam$`)`		assign names `cnam` to columns of `a`
a `= cons(`$< nr, nc, val, type >$`)`	type=	creates constant matrix
	'g'	full rectangular matrix (default)
	'd'	diagonal matrix
	'l'	lower triangular matrix
	'u'	upper triangular matrix
t `= const(`$nsiz <, val >$`)`		creates constant tensor
b `= convert(`$a, sopt$`)`	sopt=	conversion
		temperature (Fahrenheit, Celsius)
		metric $\leftrightarrow$ English
B `= cperm(`$A <, perm >$`)`		permute columns of matrix

Constants and Information (Contd.)		
Function	**Spec**	**Short Description**
d = date(< *sopt* >)	sopt=	returning date (depends on computer clock)
	missing	returns date in string form
	"ymd"	year*10000 + month*100 + day
	"year"	current year
	"month"	current month
	"day"	current day
decrypt($ofil$,$ifil$ <,pwd <,$optn$ >>)		decryption of encrypted files and directories
$ostr$ = decryp2($istr$ <,pwd <,$optn$ >>)		decryption of encrypted string objects
r = deg2rad(d)		conversion from degrees to radians
d = diag(a <,k >)		creates (sub/super) diagonal matrix
b = dia2vec(a <,k >)		moves (sub/super) diagonal from matrix to vector
i = dim(a)		returns the number of dimensions of scalar, vector, matrix, tensor
b = dimlabel(a,lab)		allocate dimension labels
lab = dimlabel(a)		fetch dimension labels
b = dimname(a,nam)		allocate dimension names
nam = dimname(a)		fetch dimension names
encrypt($ofil$,$ifil$ <,pwd <,$optn$ >>)		encryption of files and directories (incl. subdirectories)
$ostr$ = encryp2($istr$ <,pwd <,$optn$ >>)		encryption of string objects
help("*name*" <,"s" >)		opening *CMAT Reference Manual* at "name" using *Acrobat Reader*
i = ide(n)		creates $n \times n$ identity matrix
d = insert(a,b,row <,col >)		insert object b into object a
$tree$ = kdtcrt(x <,$optn$ >)		create kD tree
< $inds$,$dist$,$pnts$ > = = kdtnea($tree$,y <,$optn$ >)		find nearest neighbor nodes of kD tree
< $nfnd$,$inds$,$dist$,$pnts$ > = = kdtrng($tree$,y,$radius$ <,$optn$ >)		find nodes of kD tree in neighborhood balls
indx = loc(a <,$crit$ <,val >>)	crit	returns index locations
	"nz"	nonzeros and nonmissing values
	"ms"	missing values
	"eq"	equal to zero or value
	"ne"	not equal to zero or value
	"gt"	greater than zero or value
	"ge"	greater or equal than zero or value
	"lt"	smaller than zero or value
	"le"	smaller or equal zero or value
b = lstlabel(a,lab <,ind >)		allocate entry labels of list
lab = lstlabel(a <,ind >)		fetch entry labels of list
b = lstname(a,nam <,ind >)		allocate entry names of list
nam = lstname(a <,ind >)		fetch entry names of list

Constants and Information (Contd.)		
Function	**Spec**	**Short Description**
`macon(`$< prop >$`)`	prop=	generic and machine constants
	"nopr"	do not print table of machine constants
	"mlng"	largest long integer representation, i.e. $2^{mand} - 1$
	"mint"	largest integer representation
	"msht"	largest short integer representation
	"mand"	number of (base) digits of mantissa
	"bexp"	largest exponent before overflow
	"sexp"	smallest exponent before gradual underflow
	"ieee"	=1: computer has IEEE rounding properties, =0: no
	"base"	base of computer arithmetics
	"meps"	machine epsilon, smallest $\epsilon > 0$ with $fl(1. - \epsilon) < 1.$
	"mbig"	largest double float representation, i.e. $(base^{bexp}) * (1 - meps)$
	"msma"	smallest double float representation, i.e. $base^{(sexp-1)}$
	"savr"	value of save reciprocal, i.e. $1/savr$ does not overflow
	"l10b"	base 10 logarithm of largest double value
	"l10s"	base 10 logarithm of smallest double value
	"lnbig"	natural logarithm of largest double value
	"lnsma"	natural logarithm of smallest double value
	"pi"	$\Pi = 3.14159265358979324$
	"e"	Euler $e = 2.71828182845904523$
	"gamma"	Euler-Mascheroni $\gamma = .577215664901532861$
	"ib"	currently used index base
	"ls"	line size specification
	"ps"	page size specification
	"ifiw"	specified field width for output of integer
	"dfiw"	specified field width for output of double float
	"sing"	singularity threshold
	"spar"	range for sparsity (0 <= $spar$ <= 1)
	"relz"	relative zero threshold
to `= mat2ten(`$ml, nind <, ordr >$`)`		move list of matrices into tensor
d `= mdy(m,d,y,`$< sopt >$`)`	sopt=	obtain number of days from date input
	"year2"	two digits year input assuming 19th century date (def.)
	"y1900"	as "year2" but subtracts 1900 * 365 days
	"year4"	four digit year input
	"dm2y2"	as "year2" but day and month input is swapped
	"dm2y4"	as "year4" but day and month input is swapped

Constants and Information (Contd.)		
Function	**Spec**	**Short Description**
m = `mrand(`$kind <, a, b >$`)`	kind=	creates multivariate random matrix
	"mnor"	multivariate normal $\mathcal{N}(\mu, \Sigma)$
	"tcau"	multivariate $\sqcup(\mu, \Sigma, df)$ with $df > 0$
	"pear"	multivariate Pearson with $c \in [-1, 1]$
	"khin"	multivariate Khintchine with $df > 0$
	"mnom"	multinomial for scalar n and r vector p
	"unis"	uniformly distributed inside n dimensional sphere
	"unos"	uniformly distributed on n dimensional sphere
	"unie"	uniformly distributed inside n dimensional unit cube
	"unoe"	uniformly distributed on n dimensional unit cube
$indx$ = `order(`$vec1, vec2, \ldots$`)`		hierarchical ranking
$nxtprm$ =	sopt=	obtains n next permutations or combinations
= `permute(`$sopt, n, curind <, upprng >$`)`	"perm"	for permutations (no replications)
	"comb"	for combinations (with replications)
$next$ =		permutations and combinations
`permcomb(`$sopt, nfol, n, k <, istart >$`)`		with or without replications
id = `pid()`		returns interger ID of current process
$nams$= `prefname(`$pref, nind$`)`		returns a vector of prefix names
d = `rad2deg(`r`)`		conversion from radians to degrees
a = `rand(`$< nr, nc, type, ... >$	type=	creates random matrix
$< "dist" \mid rank, \ldots >$`)`	'g'	rectangular `rand(`$nr, nc,' r',"dist", ...$`)`
	'd'	diagonal `rand(`$n, n,' r',"dist", ...$`)`
	'u'	upper triangular `rand(`$n, n,' r',"dist", ...$`)`
	'l'	lower triangular `rand(`$n, n,' r',"dist", ...$`)`
	's'	symmetric `rand(`$n, n,' r',"dist", ...$`)`
	'r'	rectangular `rand(`$nr, nc,' r' <, rank >$`)`
	'e'	symmetric `rand(`$n, n,' e' <, rank <, elo, ehi >>$`)`
	'o'	column orthogonal `rand(`$nr, nc,' o'$`)`
	dist=	see table at *Probability Functions* below
t = `randt(`$nsiz <, dist, ... >$		creates random tensor
	dist=	see table at *Probability Functions* below
v = `randperm(`n`)`		creates a vector of the integers $1, \ldots, n$ randomly permuted

Constants and Information (Contd.)		
Function	**Spec**	**Short Description**
b = ranktie(a)		ranking entries of vector a with tied ranks averaged
$< nr, nc >$ = rccount($a <, val <, rel >>$)		count specific entries in rows and columns
d = remove(a,$inds$)		remove indexed entries from object a
b = replace($a, old, new <, rel >$)		change some values of matrix to other values
$rlab$ = rlabel(a) b= rlabel(a,$rlab$)		return row labels rlab of a assign labels rlab to rows of a
$rnam$= rname(a) b = rname(a,$rnam$)		return column labels clab of a assign names rnam to rows of a
B = rperm($A <, perm >$)		permute rows of matrix
b = shape($a <, nrb ><, ncb >$)		changes row and column number of matrix
$< c, ind >$ = setdiff(a,b)		set difference of a with b
$< c, index >$ = setisect(a,b)		set intersect of a with b
c = setmembr(a,b)		binary membership of a in b
$< c, index >$ = setunion(a,b)		set union of a and b
$< c, ind >$ = setxor(a,b)		set xor of a and b
a = size($a <, ind >$)		returns the size of the dimensions of vector, matrix, tensor
sleep($isec$)		suspend execution for isec seconds
$< ind, bmat, rank >$ = sortrow($amat$) $<, key <, optn >>$)		sorting rows of a matrix w.r.t. key formed by a set of columns
b = sortp(a,ind)		partial sorting wrt. index
irc = spawn($path, args <, optn >$)		execute child process
c = spmat($nr, nc, rind, cind, val <, sopt >$		creates sparse matrix
srand($seed <, sopt >$)		initializes uniform random generators
$< irc, str >$ = system($commands$)		execute shell command and optional save output into string object
s = time($< sopt >$)	sopt= "dtim" "hour" "min" "sec" "clock"	returning time (depends on computer clock) returns daytime hour*3600 + minute*60 + second current hour current minute current second (default) number processor clock ticks since starting job
ml = ten2mat($tens <, ordr >$)		move tensor into list of matrices
v = ten2vec($tens <, ordr >$)		move tensor to single vector
to = tenperm($tens$,$ordr$)		permutes the dimensions of a tensor
b = tri2vec($a <, opt >$)		moves (sub/super) triangle from matrix to vector
$tens$ = vec2ten(vec,$dims$)		move vector to tensor
$< c, i1, i2 >$ = unique(a)		unique entries of of a
b = vec2tri($a <, nc <, opt >>$)		moves entries of a vector into lower, upper, or symmetric matrix

6.2.4 Input and Output

In- and Output Functions	
Function	**Short Description**
a = csvread("$filepath$" <, $optn$ >)	read comma separated (CSV) values files
error($mess$)	print error message into the log
export(a,"$fname$","$ftyp$" <, $optn$ >) "ftyp" =	export (write) data set "mat", "sas", "spss", "splus", "stata", "mtb"
irc = fclose(< fid >) irc = feof(fid) irc = ferror(fid <, $iclear$ >) a = fil2obj("$filepath$")	close one or all open files test for end-of-file mark test for file read/write error reads object from file into memory
v = filestat("$filepath$") $fnam$ = fnampid("$filepath$" <,"ext" >) fid = fopen("$filepath$","$permission$") $nbyt$ = fprintf(fid,"$format$" <, a, ... >) a = fread(fid,$dtyp$ <, nr <, nc >>) irc = fremove("$filepath$") irc = frename("$oldfil$","$newfil$") a = fscanf(fid,"$format$" <, nr <, nc >>) irc = fseek(fid <, $offset$ <, $type$ >>) $nbyt$ = ftell(fid) $nbyt$ = fwrite(fid,a <, $dtyp$ >) a = generead("$filepath$",$resp$, sep <, $optn$ <, $scod$ >>)	returns a vector of file statistics returns concatenation of filename with PID open file for read/write format data for string output into file binary input from file into data remove some file rename file "oldfil" to "newfil" format string read from file into data move to byte location in open file tell current file location binary output of data to file input of comma and other separated file input of special micro array data sets
< a, b, ... > = import("$fname$","$ftyp$") "ftyp" =	import (read) data set(s) "mat", "sas", "spss", "splus", "stata", "mtb"
libname(rnk,"$dirpath$") logfile("$filepath$")	specifies or despecifies library redirects .log output to file
lstmem(< i >) lststk(< i >) lstvar(< i >)	prints table of memory allocations prints table of execution stack prints stack of symbol tables
obj2fil(a,"$filepath$") printf($format$,$arg1$, $arg2$, ...) s = pritfile("$filepath$" <, $optn$ >) rewind(< fid >)	writes object from memory to file prints formatted scalars and objects transform text file into vector of strings rewind one or all open files
a = rspfile("$filepath$" < "sep" <, $optn$ >>) sound($ifreq$ <, $octav$ <, $durat$ >>)	read sparse data set into object beep with frequ and duration
sprintf($format$,$arg1$, $arg2$, ...)	formats data into string
a = sscanf(str,$format$ <, nr <, nc >>) txtfile("$filepath$") warning($mess$)	reads scalars and objects redirects .txt output to file print warning message into the log
nr = wspfile(a,"$filepath$" < "sep" <, $optn$ >>)	writes sparse data set from object
irc = zip7("$command$") irc = zip7("$what$","$archive$" <, $file$ <, $option$ >>)	run the *7zip* program in cmat_util directory

6.2.5 Elementary Math and String Processing

Elementary Math Functions	
Function	**Short Description**
y=`abs(`z`)`	absolute value or complex modulus $\|z\|$
y=`acos(`z`)`	inverse trigonometric cosine
y=`acosh(`z`)`	inverse hyperbolic cosine
y=`asin(`z`)`	inverse trigonometric sine
y=`asinh(`z`)`	inverse hyperbolic sine
y=`atan(`z`)`	inverse trigonometric tangent
y=`atan2(`x,y`)`	trigonometric tangent
y=`atanh(`z`)`	inverse hyperbolic tangent
y=`ceil(`z`)`	rounds to an integer upward to $+\infty$
y=`cos(`z`)`	trigonometric cosine
y=`cosh(`z`)`	hyperbolic cosine function
y=`exp(`z`)`	exponential function
y=`fix(`z`)`	rounds to an integer toward 0
y=`floor(`z`)`	rounds to an integer downward to $-\infty$
r=`fmod(`x,y`)`	remainder r of $x = ky + r$
y=`ldexp(`x,i`)`	returns xj^i for $radix\, j = 2$
y=`log(`z`)`	natural logarithm function
y=`log2(`z`)`	base-2 logarithm
y=`log10(`z`)`	base-10 logarithm function
y=`max(`$a <, b >$`)`	maximum in a or between a and b
$< y, ind >$=`max(`$a, b, ...$`)`	maximum value and index location
$< y, ind >$=`maxn(`a, n`)`	n largest values and index locations
y=`min(`$a <, b >$`)`	minimum in a or between a and b
$< y, ind >$=`min(`$a, b, ...$`)`	minimum value and index location
$< y, ind >$=`minn(`a, n`)`	n smallest values and index locations
y=`pow(`z,c`)`	power function z^c
i=`rand()`	pseudorandom generator
y=`round(`$z <, ndig >$`)`	rounds toward the nearest integer
y=`sign(`z`)`	signum function ($y = 0$ for $x = 0; y = 1$ for $x > 0; y = -1$ for $x < 0$)
y=`sign2(`a,b`)`	signum function ($y = \|a\|$ for $b \geq 0; y = -\|a\|$ for $b < 0$)
y=`sign4(`$a1, a2, a3, b$`)`	signum function ($y = a1$ for $b < 0; y = a2$ for $b = 0; y = a3$ for $b > 0$)
y=`sin(`z`)`	trigonometric sine function
y=`sinh(`z`)`	hyperbolic sine function
i=`sizeof(`z`)`	number of bytes used for the storage of z
y=`sqrt(`z`)`	square root $\sqrt{z}$
y=`tan(`z`)`	trigonometric tangent function
y=`tanh(`z`)`	hyperbolic tangent function

String Processing Functions	
Function	**Short Description**
int =atoi(*s*)	converts string `s` to integer
real=atof(*s*)	converts string `s` to float
b = byte(*a*)	transfers integers (0,..,255) into ASCI chars
b = noblanks(*a* <, *str* >)	removes leading and trailing blanks from string data
str = strcat(*s*1, *s*2 <, *n* >)	concatenate `s2` to the end of `s1`
str = strchr(*s*, *c*)	search `s` for first occurence of `c`
int = strcmp(*s*1, *s*2 <, *n* >)	compare `s1` with `s2` lexicographically
str = strcpy(*s*1, *s*2 <, *n* >)	overwriting copy of `s2` to `s1`
int = strcspn(*s*, *set*)	search `s` for first occurence of char included in `set`
int = strlen(*s*)	return length of string `s`
str = strlwr(*s*)	convert string to lower case
int = strpos(*s*, *c*)	search `s` for first occurence of `c`
str = strrchr(*s*, *c*)	search `s` for last occurence of `c`
str = strrev(*s*)	reverse string
int = strrpos(*s*, *c*)	search `s` for last occurence of `c`
int = strspn(*s*, *set*)	search `s` for first occurence of char **not** included in `set`
str = strupr(*s*)	convert string to upper case

Note, that all `str...()` functions have been extended to vector, matrix, and tensor arguments.

6.2.6 Advanced Math

Advanced Math Functions		
Function	**Spec**	**Short Description**
v = `besseli(`$x <, alpha <,$"esc" $>>$`)`		Bessel I function
v = `besselj(`$x <, alpha <,$"esc" $>>$`)`		Bessel J function
v = `bessely(`$x <, alpha <,$"esc" $>>$`)`		Bessel Y function
v = `besselk(`$x <, alpha <,$"esc" $>>$`)`		Bessel K function
y = `dawson(`x`)`		Dawson integral
v = `ellipi(`$kind, x <, y >$`)`		complete elliptic integral
v = `ellinc(`$kind, x, y, z <, \rho >$`)`		incomplete elliptic integral
y = `expint(`$x <,$"esc"|"one" $>$`)`		exponential integral
y = `factrl(`$a <, b <, c >>$`)`		factorial
v = `ferdirc(`ord, x`)`		Ferri-Dirac integral
v = `ferdiri(`ord, x, b`)`		incomplete Ferri-Dirac integral
y = `fft(`$x <, optn >$`)`		fast Fourier transform
y = `ffti(`$x <, optn >$`)`		inverse fast Fourier transform
$< g, j, c, h >$ = `fider(`$f, x <, sopt <, par <, grad >>>>$`)`	sopt =	derivatives by finite differences
	"grad"	returns gradient
	"jaco"	returns Jacobian
	"crpj"	returns cross product Jacobian
	"hess"	returns Hessian
	"forw"	uses forward difference formula
	"cent"	uses central difference formula
	"expo"	uses Richardson extrapolation
	"fsum"	applies on sum of functions
	"flsq"	applies on sum of squares of functions
$xmin$ = `fmin(`$func, xest$	sopt=	minimization of univariate function
$<, range <, sopt <, par <, grad >>>>$`)`	"bre"	method by Brent
z = `fresnel(`x`)`		Fresnel integral
$zero$ = `fzero(`$func, xest$	sopt=	zero of univariate function
$<, range <, sopt <, par <, grad >>>>$`)`	"bre"	method by Brent
	"b+d"	method by Bus and Dekker
	"mul"	method by Muller
y = `gcd(`a, b`)`		greatest common divisor
$< gof, circ >$ = `hamilton(`$indx, vert <, optn >$`)`		find (all or some) Hamilton circuits
$< gof, circ >$ = `hamilton(`$adja <, optn >$`)`		
y = `horner(`$coef, xval <, opt >$`)`		evaluate the Horner scheme
$< f1, d1, d0 >$ = `intpol(`$x0, f0, x1 <, sopt >$`)`	sopt=	univariate interpolation
	"lin"	linear
	"cub"	cubic
	"spl"	spline

Advanced Math Functions (Contd.)		
Function	**Spec**	**Short Description**
$< gof, xind >$ = knapsack($prof, wgt, cap <, optn >$)		one- and multi-dimensional Knapsack problem
res = latlong($task, pnt1 <, pnt2 <, optn >>$)		functions of Latitude and Longitude data
$< gof, post >$ =		multifacility location problem
= locat1($vwgt, fwgt <, optn >$)		with rectilinear distance
$< gof, xind, yind, ofun, lm >$ =		assigning K optimal locations among $n > K$
= locatn($cmat <, par <, dvec >>$)		potential locations for servicing m clients.
$< x, lm, rp, duals, sens >$ =	meth	minimize or maximize linear function
= lp("$meth$", $c, lau <, lbub <, optn <, xint >>>$)		
= lp("$meth$","$path$" $<, optn <, xint >>>$)	"lps"	using lpsolve() (Berkelaar et al., 2004)
	"pcx"	using interior point PCx algorithm (Czyzyk et al, 1997)
	"con"	using LPASL continuation method (Madsen & Pinar, 1993)
	"clp"	using Clp code by J. J. Forrest
$< x, rp, duals, sens >$ = lpassign($cost <, optn >$)		linear assignment problem
$< x, rp, duals, sens >$ =		linear transportation problem
= lptransp($cost, rrhs, crhs <, optn >$)		
$< c, lau, lubc >$ =		reading MPS file for LP
= mpsread($fpath <, optn >$)		
$< prob, row, col, rhs, rng, bnd >$ =		writing MPS file for LP
= mpswrite($strvec, c <, lau <, lubc >>$)		
$< ilnk, dlnk >$ = mstdis($x <,$"$meth$" $<, optn >>$)	"meth"	minimum (maximum) spanning tree based on distances
	"rohlf"	method by Rohlf (1978)
	"whitn"	method by Whitney
$< gof, pred >$ = mstgra($indx, vert, cost <, optn >$)		find minimum spanning tree based on graph data
$< gof, pred >$ = mstgra($adja, cost <, optn >$)		

Advanced Math Functions (Contd.)		
Function	**Spec**	**Short Description**
$< xr, rp, der1 >$ `= nle(`$func, x0$	tech=	solve system of nonlinear equations
$<, sopt <, par, < jac >>>$`)`		for options settings see reference manual
$< xr, rp, der1, der2, acon, dpro, jhnlc >$	tech=	minimize or maximize nonlinear function
`= nlp(`$func, x0 <, optn <, lbub <, lau$	"QADPEN"	Quadratic Penalty Algorithm
$<, nlcon <, grad <, hess < jcon >>>>>>>$`)`	"DQNSQP"	Quasi-Newton SQP Method
	"VMCWD"	Powell's original VMCWD method
	"TRUREG"	Trust-Region Method
	"NEWRAP"	Line Search Newton-Raphson Method
	"NRRIDG"	Ridge Newton-Raphson Method
	"DBLDOG"	Double-Dogleg Methods (DBFGS,DDFP)
	"QUANEW"	Quasi-Newton Methods (DBFGS,DDFP,BFGS,DFP)
	"POWBLC"	Powell's BFGS for Linear Constraints
	"LMQUAN"	Limited Memory Quasi-Newton Methods
	"CONGRA"	Conjugate Gradient Methods
		with versions: PB, FR, PR, CD, BM, SFR, SPR
	"NONDIF"	Nondifferential Subgradient Methods (BT)
		with different versions
	"NMSIMP"	Nelder-Mead Simplex Method
	"COBYLA"	Constrained Optimization by Linear Approximation
	"LINCOA"	Linear constrained Optimization Algorithm
	"BOBYQA"	Bound constrained Optimization by
		Quadratic Approximation
	"UOBYQA"	Unconstrained Optimization by
		Quadratic Approximation (default version=1)
		version=0: "NEWUOA" modified "UOBYQA"
	"SIMANN"	Simulated Annealing (Global Opt.)
	"GENALG"	Genetic Algorithm (only maximization)
	"LEVMAR"	Levenberg-Marquardt Method
	"HYQUAN"	Hybrid Quasi-Newton Methods (DBFGS, DDFP)
	"NL1REG"	Nonlinear L_1 Regression
	"NLIREG"	Nonlinear L_∞ Regression
	"MINMAX"	Nonlinear MinMax Optimization
	"NONE"	do not perform any optimization
$< y, t >$ `= ode(`$func, tab, y0$	optn=	ordinary differential equations
$<, par <, root <, afun <, jacm >>>$`)`	par[4]=1	Adams method (nonstiff)
	par[4]=2	Gear's method (stiff)
	par[4]=3	switch dynamically Gear's or Adams
	par[5]=0	$\frac{dy_i}{dt} = f_i(y,t), \quad i = 1, \ldots, n$
	par[5]=1	$\mathbf{A}\frac{dy}{dt} = f(y,t)$
	par[5]=2	$\mathbf{a_i}\frac{dy_i}{dt} = f_i(y,t),\ i = 1, \ldots, m$
		$0 = f_i(y,t), \quad i = m+1, \ldots, n$

Advanced Math Functions (Contd.)		
Function	**Spec**	**Short Description**
$< xr, lm, rp >$ = pcx("$mpsfil$" $<, keyopt >$)		PCx algorithm for linear programming
$< coef, sse >$ = polyfit($x, y, d <, opt >$)		fit polynomials of degree d
z = polynom($a <, sopt <, x >>$)	sopt	operations on polynomial
	"zero"	find zeros
	"eval"	evaluate polynomial
	"deri"	evaluate first derivative
	"coef"	return coefficients for given zeros
y = polyval($coef, x <, opt >$)		evaluate polynomials at x
$< b, e >$ = primfact(x)		prime factors of x
$< xr, rp >$ = qp($mat <, vec$	tech=	minimize or maximize quadratic function
$<, lau, <, x0 <, optn <, lbub >>>>>$)	"QPNUSP"	Null Space (Active Set) Method
	"QPRASP"	Range Space (Active Set) Method
	"QPPOGI"	Goldfarb and Idnani Algorithm by Powell
	"QPTRON"	Lin-Moré Trust-Region (TRON) Method (only BC)
	"QPMANP"	Madsen-Nielsen-Pinar Algorithm (only BC)
	"QPBARR"	Interior Point (Barrier) Algorithm (only BC)
$< H, c, y >$ = qptst($n <, par >$)		create Moré-Toraldo QP test problem
$area$ = quad($func, ab <, optn >$)		quadrature of function
		for options settings see reference manual
$< area, d0 >$ = quadip($x0, f0, ab <, sopt >$)	sopt	quadrature of interpolation
	"lin"	linear
	"cub"	cubic
	"spl"	spline
$< area, aerr, nfun >$ =		quadrature with Gaussian weights
quadirgw($func, ndim <, optn >$)		over infinite region
		adaptive and stochastic method
$< area, aerr, nfun >$ =		multivariate vector quadrature
quadsimp($func, vertx <, optn >$)		over simplex region
$< area, aerr, nfun >$ =		multivariate vector quadrature
= quadvec($func, ab <, optn >$)		over rectangular region
		adaptive and stochastic method
$< gof, path, len >$ =		find shortest path length between
= splnet($indx, vert, cost, brack <, optn >$)		two nodes of a network
$< gof, path, len >$ = splnet($adja, cost, brack <, optn >$)		
r = ssq(v)		sum of squares of entries
$< length, tour, dist >$ =		traveling salesman problem
= tsp($meth, norm, data <, optn <, intour >>$)		

Matrix Functions		
Function	**Spec**	**Short Description**
$< val, vec, res, scl >=$	sopt	ARPACK Functions: Matrix spec.
`=` `arpack(`$a, sopt, nv, ncv <, optn <, b >>$`)`	"sev","gsev"	EVD: real symmetric
	"nev","gnev"	EVD: real unsymmetric
	"zev","gzev"	EVD: complex unsymmetric
$< val, lvec, rvec, res, scl >=$	sopt	ARPACK Functions: Matrix spec.
`=` `arpack(`$a, sopt, nv, ncv <, optn >$`)`	"svd"	SVD: selected values (vectors)
$< val, vec, res, scl >$ `=`	sopt	ARPACK Functions
`=` `arpack(`$op\_fun, sopt, nv, ncv <, optn >$`)`		operator function specification
$< val, lvec, rvec, res, scl >$ `=`	sopt	ARPACK Functions
`=` `arpack(`$op\_fun, sopt, nv, ncv <, optn >$`)`	"svd"	SVD operator function specification
$< l, p, r, rcon >$ `=` `chold(`$a, sopt$`)`	sopt	Cholesky decomposition
	"piv"	perform pivoting
	"eng"	use Ng-Peyton-Liu sparse minimum degree ordering
	"add"	use Amestoy-Davis-Duff sparse minimum degree ordering
	"gmw"	Gill-Murray-Wright modified Cholesky D.
	"esc"	Eskow-Schnabel modified Cholesky D.
	"mor"	Morè modified Cholesky Decomposition
$< q, r, v, pi, lind >$ `=` `cod(`$a <, b <, sopt >>$`)`		complete orthogonal decomposition
$< x, lind >$ `=` `codapp(`$a, b <, sopt >$`)`		minimum length solution of rank deficient LSQ
r `=` `cond(`$a <, sopt >$`)`	sopt	condition of matrix
	"svd"	use singular value decomposition
	"est"	use iterative estimation
b `=` `cumprd(`$a <, sopt >$`)`	sopt	cumulative product
	"lr"	left-right
	"ud"	upper-down
b `=` `cumsum(`$a <, sopt >$`)`	sopt	cumulative sum
	"lr"	left-right
	"ud"	upper-down
r `=` `det(`a`)`		determinant of matrix
d `=` `design(`$v <, opt >$`)`		(stat.) design matrix
d `=` `diag(`$a <, k >$`)`		creates (sub/super) diagonal matrix
v `=` `dia2vec(`$a <, k >$`)`		moves (sub/super) diagonal from matrix to vector
b `=` `echelon(`a`)`		row echelon normal form
$< eval, rvec, lvec >$ `=` `eig(`a	sopt1	eigenvalues and eigenvectors
$<, sopt1 <, sopt2 <, vl <, vu >>>>$`)`	"impql"	implicit QL method
	"bisec"	bisection method
	"ratql"	rational QL method
	"jacob"	Jacobian method
	"dicon"	divide-and-conquer
	sopt2	
	"nobal"	do not scale nor permute
	"nosca"	do not scale
	"noper"	do not permute

Matrix Functions (Contd.)		
Function	**Spec**	**Short Description**
$< gof, Aica, Wica, Scor, Eval, Evec >$ = = fica($data, optn <, Eini <, Vini <, Agues >>>$)		(fast) independent component analysis
b = flip($a <, sopt >$)	sopt "lr" "ud"	flip entries in matrix left-right upside-down
$< eval, rvec, lvec >$ = geig($a, b <, type >$)	type= 1 2 3	generalized eigenvalue problem $\mathbf{AZ} = \mathbf{\Lambda BZ}$ $\mathbf{ABZ} = \mathbf{\Lambda Z}$ $\mathbf{BAZ} = \mathbf{\Lambda Z}$
$< x, y >$ = glm(a, b, d)		$\min_x \parallel y \parallel_2$ subject to $\mathbf{A}x + \mathbf{B}y = d$
$< eval, form_a, form_b, rvec, lvec >$ = = gschur(a, b)		generalized Schur decomposition
$< sig_a, sig_b, v, r, u_a, u_b >$ = gsvd(a, b)		generalized singular value decomposition
c = hankel($a <, p >$)		create Hankel matrix
c = hdprod(a, b)		horizontal direct product
u = hhold($v <, ind >$)		compute Householder transform
b = hnform(a)		Hermite normal form
b = inv(a)		inverse of matrix
b = invupd($a, v \vert\vert s$) c = invupd(a, v, s)		update of inverse matrix update of inverse matrix
$< x, norm >$ = ldp($a, b <, optn >$)	sopt=	linear distance programming
$< x, nrnk, (sval \vert rcon) >$ = lls($a, b <, sopt <, optn >>$)	sopt= "svd" "cod" "qrd" "lqd" "evd" "gel" "itr"	$\min_x \parallel \mathbf{A}x_j - b_j \parallel_2$, $j = 1, \ldots, p$ use singular value decomposition use complete orthogonal decompostion use QR decomposition use LQ decomposition use eigenvalue decomposition (symmetric $\mathbf{A}$) use Gaussian elimination with normal equations use iterative method specified by optn e.g. LSQR by Paige and Saunders; CGNR, CGNE
x = lsolv($a, b <, sopt <, optn >>$)	sopt= "svd" "cod" "qrd" "lqd" "evd" "gel" "itr"	$\min_x \parallel \mathbf{A}x_j - b_j \parallel_2$, $j = 1, \ldots, p$ use singular value decomposition use complete orthogonal decompostion use QR decomposition use LQ decomposition use eigenvalue decomposition (symmetric $\mathbf{A}$) use Gaussian elimination (sparse or dense) use iterative method specified by optn SYMMLQ, CG, MINRES; LSQR; CGsqu, BiCG, BICGstab, CGNR, QMR, GMRES
x = lse(a, b, c, d)		$\min_x \parallel \mathbf{A}x - c \parallel_2$ subject to $\mathbf{B}x = d$
$< l, u, pi >$ = lud(a)		LU decomposition

Matrix Functions (Contd.)		
Function	**Spec**	**Short Description**
$< dist, ldet, rmu, rcov >$ = = mahalanobis($x <, mu <, cov <, optn >>>$)		Mahalanobis distance (whole matrix or only diagonal)
i = ncol(a)		number of columns of matrix
$< w, h >$ = nnmf($a, k < optn <, uini <, wini >>>$) $< w, h, d >$ = nnmf($a, k < optn <, uini <, wini >>>$)		nonnegative matrix factorization
r = norm($a <, p >$)		vector or matrix norm
i = nrow(a)		number of rows of matrix
z = nullsp($x <, sopt <, tol >>$)	sopt= "svd" "qrd"	null space of matrix use singular value decomposition use QR decomposition
$coeff$ = ortpol($a <, maxdeg < wgt >>$)		coefficients of orthogonal polynomials
$< u, r, v >$ = ortvec($a <, q >$)		compute vector orthogonal to columns
$< b, nrnk, sval \vert rcon >$ = pinv($a <, sopt >$)	sopt= "svd" "cod" "evd"	use singular value decomposition use complete orthogonal decomposition use eigenvalue decomposition
n = profile(a) $< b, perm >$ = profred(a)		computes profile of matrix reduces profile of matrix
$< q, r, pi >$ = qrd($a <, ind >$)		QR decomposition
z = rangesp($x <, sopt <, tol >>$)	sopt= "svd" "qrd"	range space of matrix use singular value decomposition use QR decomposition
z = rank($a <, sopt <, tol >>$)	sopt=	rank of matrix
	"svd" "qrd"	use (dense) singular value decomposition use (dense or sparse) QR decomposition
r = rcond($a <, sopt >$)	sopt= "svd" "est"	reciprocal condition of matrix use singular value decomposition use iterative estimation
$< form_t, vec_z, eval >$ = schur(a)		Schur decomposition
$< gof, D, X, Y, Rho >$ = sdd($data, nfac <, optn <, wgt >>$)		semi discrete decomposition
$< sval, v, u >$ = svd($a <, p \vert$"eco" $<, optn >>$)	sopt= "eco"	singular value decomposition economic version (small **U** or **V**)
$< s, v, u >$ = svdtrip(a,"$meth$" $<, optn >$) $< s, v, u >$ = svdtrip($funa$,"$meth$" $<, optn <, funata >>$)	"meth" "bls","las",... "meth" "bls","las",...	compute (s, v, u) triplets of largest singular singular values and vectors compute (s, v, u) triplets of largest singular singular values and vectors
$< s2, v2, u2 >$ = svdupd($a, b <, s0, v0, u0 >$)		rank r update of SVD
b = sweep($a <, p >$)		sweep matrix
z = sylve($a, b, c <, s >$)		solve Sylvester equation
$< y, perm >$ = tarjan(x)		permute row/col Tarjan's algorithm

Matrix Functions (Contd.)		
Function	**Spec**	**Short Description**
$tens\|scal\|vec$ = tentvec($tens, vec <, n >$)		multiply tensor with vector
tens = tentmat($tens, mat <, n >$)		multiply tensor with matrix
$tens\|scal$ = tentten($aten, bten <, n >$)		multiply tensor with tensor
c = toeplitz(a)		create Toeplitz matrix
b = toeplitz(a,"dur")		solve Yule-Walker equations using Durbin algorithm
b = toeplitz(a,"lev",b)		linear system with Toeplitz matrix Levinson algorithm
b = toeplitz(a,"tre")		invert Toeplitz matrix using Trench algorithm
r = trace(a)		trace of matrix
$< b, c, s >$ = tridod(a, v)		Rank-1 downdate of Cholesky factor
$< b, c, s >$ = triupd(a, v)		Rank-1 update of Cholesky factor

6.2.7 Statistics

Statistical Functions		
Function	**Spec**	**Short Description**
$< powr, gvec, gmat >$ = acss($stim, dgrp$ $<, optn <, nisac >>>$)		animal carcinogenicity power and sample size Peto test, power computed by Weibull MC
$< scor, vars, rprm, cprm, sprm >$ = anacor($a, optn$)		correspondence analysis of contingency table
$< scor, vars >$ = anaprof($a, optn$)		correspondence analysis of profile data
$< auc, tab, xopt, cov >$ = = auroc($data <, optn <, popt <, x0 > . >$)		area under the receiver operating curve with as. standard error
$< gof, coef, covm, pred >$ = = bidimreg($type, y, x, <, optn >$)		bidimensional regression
$< c, ase, conf, acov >$ = bivar($a <, sopt1 <, sopt2 <, ipar >>>$)	sopt1= "scp" "cov" "corr" "spea" sopt2= "par" "inv" "pinv"	bivariate functions SSCP (scalar product) matrix covariance matrix Pearson correlation matrix Spearman correlation matrix partial correlation inverse of covariance matrix pseudoinverse of covariance matrix
$< gof, bstv, stat, hist, his2 >$ = boruta($data, modl <, optn <, class <, cwgt >$)		Boruta (feature selection) algorithm (based on Random Forest modeling)
$< gof, eval, xload, yload, xscor, yscor, xcqua, ycqua >$ = canals($xydata, scale, xind, yind, optn$)		canonical correlation (nonmetric Gifi version)
$< gof, sqcc, cstd, craw, swth, sbtw >$ = cancor($data, optn, xind, yind$)		canonical correlation
$< u, v, d >$ = centroid($a, k < optn >$)		centroid factorization
$< gof, est, resi, cov, mod1, mod2, boci >$ = = cfa($data, optn, <, patt <, scal <, wgt <, xini <, targ <, wtrg > . >$)		confirmatory factor analysis
$< memb, crit, mu, sigma, pi, asemu, asesig, asepi >$ = = clmix($x, optn <, par3 <, par4 <, par5 >>>$)		Fitting mixtures of normal and t components

Statistical Functions (Contd.)		
Function	**Spec**	**Short Description**
$< est, serr, conf, pval >$ `=` `conting(`$a, sopt <, par >$`)`	sopt=	association of contingency tables
	"corr"	Pearson correlation
	"spea"	Spearman rank correlation
	"polchor"	polychoric correlations
	"gamma"	Goodman-Kruskal gamma
	"taub"	Kendall's tau_b
	"tauc"	Kendall's tau_c
	"somcr"	Somer CR
	"somrc"	Somer RC
	"lambcr"	lambda CR
	"lambrc"	lambda RC
	"lambsym"	symmetric lambda
	"unccr"	uncertainty CR
	"uncrc"	uncertainty RC
	"uncsym"	symmetric Uncertainty
	"exrisk"	exact risk
	"oddrat"	odds ratio
$< cont, ntot >$ `=` `contsim(`$rsum <, csum <, ntab >>$`)`		simulate contingency table with specified row and column sums
$< cov, dlta, fcov, scov, smu >$ `=` `covshr(`$xdat, sopt, <, par <, rind >>$`)`	sopt=	shrink covariance matrix
	"corr"	average correlation method
	"mark"	market return method
	"diag"	diagonalmethod
	"twop"	two parameter method
$< est, ecov, ocov >$ `=` `delta(`$xstar, fest, fopt <, opt <, egrd >>$`)`		Delta method
$< gof, estim, yxprd >$ `=` `demreg(`$data <, opt >$`)`		(univar.) Deming regression
d `=` `dist(`$x <, sopt <, optn <, scal >>>$`)`	sopt=	distance matrix
	"L2"	Euclidean (L_2) Distances
	"L1"	City-Block (L_1) Distances
	"Li"	Maximum (L_∞) Distances
	"Gower"	Gower Dissimilarities
$< cov, mu, b >$ `=` `emcov(`$a <, par >$`)`		estimate covariance matrix and mean vector when data have missing values

Statistical Functions (Contd.)		
Function	**Spec**	**Short Description**
$< y, t, c >$ =	sopt	rotate factors to simple structure
= `frotate(`$x <, sopt <, par <, targ <, weig >>>>$`)`	"crafer"	Crawford-Ferguson family
	"varmax"	Varimax rotation
	"quamax"	Quartimax rotation
	"equmax"	Equamax rotation
	"parmax"	Parsimax rotation
	"facpar"	Factor Parsimony rotation
	"bentlr"	Bentler rotation criterion
	"minent"	Minimum Entropy rotation
	"tandm1"	orthogonal Tandem 1 rotation
	"tandm2"	orthogonal Tandem 2 rotation
	"infmax"	orthogonal Infomax rotation
	"mccamm"	orthogonal McCammon rotation
	"oblmin"	Direct Oblimin family
	"quamin"	Direct Quartimin
	"biqmin"	Bi-Quartimin ritation
	"covmin"	Covarimin rotation
	"simmax"	Simplimax oblique rotation
	"oblmax"	Oblimax rotation
	"geomin"	Geomin rotation
	"promax"	Promax oblique rotation
	"targt1"	Target rotation
	"targt2"	(partially specified) Target rotation
$< gov, est, resi, cov >$		factor analysis (exploratory)
= `factor(`$data, optn <, wgt <, init <, prior >>>$`)`		
$< beta, shrnk, yptrn, yptst >$		Garotte (Breiman, 1993)
= `garotte(`$trn, model, optn <, class <, test >>$`)`		nonnegative linear regression

Statistical Functions (Contd.)		
Function	**Spec**	**Short Description**
$< gof, parm, ncov, rcov, yhat >$ = = `gee`$(data, model <, optn, <, class <, rstr <, xini >>>)$		generalized estimation of equations
$< gof, parm, ase, conf, cov, typ1, typ3, yhat, roc >$ = `glim`$(data, model <, optn, <, class <, xini <, contr >>>>)$		generalized linear models
$< gof, parm, ase, conf, cov, typ1, typ3, yhat, theta, roc >$ = `glmixd`$(data, model <, optn <, class <, random <, xini <, contr >>>>)$		mixed generalized linear models (random effects; ordinal and nominal response)
$< gof, parm, ase, conf, cov, typ1, typ3, resi >$ = `glmod`$(data, model <, optn, <, class <, cont >>>)$		general linear model
gof = `hbaddtst`$(data, < optn >)$		Test for additivity by Tukey (Nollau, 1975)
$< gof, hyp, ci >$ = `hbanova`$(data <, freq <, optn >>)$		Nine ANOVA models I and II (Nollau, 1975)
gof = `hbbartlett`$(data <, freq <, optn >>)$		Bartlett test (Nollau, 1975)
gof = `hbcovar`$(ydata, wdata <, freq <, optn >>)$		two models for covariance analysis with one covariable discriminance analysis (Nollau, 1975)
$< gof, fsep, yhat, zhat >$ = = `hbdiscrim`$(data, idc <, optn <, zdat >>)$		
$< gof, parm, yhat, zhat, test >$ = = `hblreg`$(xdat, ydat <, optn <, zdat <, cont >>>)$		linear least squares regression (Nollau, 1975)
gof = `hbltst`$(xdat, ydat, freq <, optn >)$		linearity test by R. A. Fisher (Nollau, 1975)
$< gof, parm, yhat >$ = = `hblqrg`$(xdat, ydat, mpow, freq, <, optn >)$		Simple polynomial regression (Nollau, 1975)
$< gof, parm >$ = `hbrcmp`$(x1, y1, x2, y2 <, optn >)$		Compare coefficients of two linear regressions
gof = `hbscheffe`$(data, cont <, freq <, optn >>)$		Scheffè (1959) test (Nollau, 1975).
$< ttest, mom, ftest >$ = `hbttest`$(data1, data2 <, optn >)$		uni- and bivariate t test and Welch's test
$hist$ = `histogrm`$(a, k, <, optn >)$		obtain k histogram
$< gof, eval, discr, obscor, cquant, stquant >$ = `homals`$(a, optn)$		homogeneity analysis of multimonial data
$< gof, parm >$ = `hotell`$(x, mu0 <, optn >)$		standard/robust one-sample Hotellings Test
`yhat = isoreg`$(yxw <, optn >)$		(weighted) isotone regression
$< gof, est, cov >$ = `irtml`$(data, optn <, tini <, bini <, bc >>>)$		Maximum Likelihood IRT (various algorithms)
$< gof, scal, loev, pairs > =$ = `irtms`$(data, optn)$		(nonparameteric) Mokken scale IRT by Hardouin (2007)

Statistical Functions (Contd.)		
Function	**Spec**	**Short Description**
$< jack, boot >$ =	task=	jackknife and bootstrap
= `jboot(`$data$,"$task$", $usfun <, optn >$`)`	"all"	perform all methods
	"jack"	Jackknife
	"norm"	normal
	"perc"	percentile
	"hybr"	hybrid
	"bc"	bias corrected
	"bca"	bias corrected accelerated
	"stud"	Studentized, bootstrap-t
$< gof, dens, mesh >$ =		1- and 2-dimensional Gaussian
= `kde(`$data, optn$`)`		kernel density estimation
$< gof, beta, yptrn, yptst >$		LARS and related methods
= `lars(`$trn, model, optn <, class <, test >>$`)`	type=	L_1 and L_2 constrained linear regression
	1: lars	least angle reagression (Tibshirani et al., 1996)
	2: lasso	Lasso (default) (Tibshirani et al., 1996)
	3: stage	forward stagewise (Tibshirani et al., 1996)
	4: rdige	Ridge Regression;
	5: lars-en	Elastic Net (Zou & Hastie, 2003);
	6: ust	Univariate Soft Thresholding (Donoho et al., 1995)
$< coef, null >$ = `lda(`$sym <, par >$`)`		analysis of linear dependencies
$< gof, best, parm, yptr, yptt >$ = `lrallv(`$msel, trn,$		all variable subset linear regression
$model, optn <, class <, aov16 <, grpv <, tst >>>>$`)`		
$< gof, parm, yptr, yptt >$ = `lrforw(`$trn,$		stepwise forward linear regression
$model, optn <, class <, aov16 <, grpv <, tst >>>>$`)`		variable selection for large applications
v = `mad(`$a <, optn >$`)`		median absolute deviation (MAD)
$parm$ = `mardia(`$a <, sopt <, optn >>$`)`	sopt=	mv skewness and kurtosis with p values
	"mvk"	multivariate kurtosis
	"mvs"	multivariate skewness
	"all"	multivariate kurtosis and kurtosis
$< gof, conf, wgt, ... >$ =	meth	multidimensional scaling
= `mds(`$data, optn <, xini >$`)`	"kyst"	Kruskal-Young-Shepard-Torgerson
	"smacof"	method by de Leeuw and Heiser
	"multsc"	ML method by Ramsay (1977)
	"indscal"	method by Carroll and Chang (1970)
	"sumscal"	method by de Leeuw and Pruzansky (1978)
	"cospa"	method by Schoenemann (1972)
v = `median(`$a <, optn >$`)`		median
$< gof, parm, cov, resi >$ =		mixed effects
= `mixregv(`$data, modl, optn, covar <, rand <, errvar >>$`)`		location-scale regression
$< x, r, c >$ = `mpolish(`$a <, optn >$`)`		mean and median polish
$< gof, lik, est, fpbar, BF >$ = `mucomp(...)`		test group and order restrictions on means
$< gof, BF >$ = `mucomp(`$data, nres, modl <, optn >$`)`		(confirmatory ANOVA)
$< gof, stat, parm >$ =		multiple comparison of $K > 2$ means
= `multcomp(`$imet, g, x <, optn <, con <, nams >>>$`)`		

Statistical Functions (Contd.)		
Function	**Spec**	**Short Description**
$< prob, stat >$ `= mvntest(`$sopt, x <, optn >$`)`	sopt=	testing for multivariate normality
	"mark"	Mardia's test for multivariate kurtosis
	"mard"	Mardia's test for multivariate skewness
	"maro"	Mardia & Foster (1983) omnibus test
	"wmin"	Wang and Hwang (2011) test
	"roys"	Royston (1983) W test
	"hezi"	Henze-Zirkler (1990) test
	"q123"	Small's $Q1$, $Q2$, $Q3$
	"doha"	Doornik-Hansen Omnibus test
	"szek"	Szekely-Rizzo (2005) test
	"mudh"	Mudholkar-McDermott-Srivastava
$< gof, parm, ase, conf, cov >$ `= nlreg(`$func, x0$	optn	nonlinear L_p norm regression
$<, optn <, nlpopt <, lbub <, lau <, nlcon$		with Wald, PL, and Jackknife CI's
$<, grad <, hess <, jcon > .. >$`)`	L_2	least squares
	L_1	L_1 or LAV regression
	L_∞	Chebychev regression (no CI's)
	L_p	general L_P with $1 \leq p \leq 10$
$model, optn <, class <, tun <, kfun <, test >>$`)`		
$< gof, est, resi, cov >$		binary factor analysis
`= noharm(`$data, optn <, guess <, init >>$`)`		
$< parm, delt, ase, conf, cov >$ `= odr(`$func, x0, xdat$		nonlinear orthogonal distance regression
$<, ydat <, optn <, fjac <, d0$		(incl. OLS regr. w. mult. response)
$<, we <, wd <, bfix <, xfix > ... >$`)`		
$< oind, oval, crit >$ `= outlier(`$a, sopt <, optn >>$`)`	sopt=	univariate outlier detection
	"chisq"	χ^2 method
	"sig_3"	3-Sigma rule
	"tukey"	Tukey method
	"chauv"	Chauvenet method
	"grubb"	Grubbs (1969) test
	"thomp"	Thompson τ test
	"dixon"	Dixon Q test
$< mkur, madi >$ `= outlmd(`$a <, sopt1 <, opt1 >>$`)`	sopt1=	multidimensional outlier detection
	"mvk"	multivariate kurtosis
$< loc, scat, dist >$ `= outlmd(`$a <, sopt2 <, opt2 >>$`)`	sopt2=	multidimensional outlier detection
	"mve"	minimum volume ellipsoid
	"mcd"	minimum covariance determinant
$< gof, eval, cquan, load, scor >$ `=`		K set canonical correlation analysis
`= overals(`$data, vtyp, sets, optn$`)`		(Gifi version)
$< psym, pind >$ `= partial(`$asym, ipar <, par >$`)`		partial covariances or correlations
$< gof, eval, evec, comp >$ `=`		principal component analysis (PCA)
`= pca(`$data, optn <, targ <, wgt >>$`)`		(various column and row oriented algorithms)
$< rms, parm, yprd, vrms, trms, tprd >$ `=`		partial least squares (PLS)
`= pls(`$xtrn, yind, xind <, icmp <, optn <, xtst >>>$`)`		and principal components regression (PCR)
$< corr, tau, ccov, tcov, ctco, gof >$ `=`		polychoric correlations with
`= polychor(`$data <, ind <, optn >>$`)`		thresholds and asymptotic covariances

Statistical Functions (Contd.)		
Function	**Spec**	**Short Description**
$< gof, eval, obscor, cquant, loada, scora, loadb, scorb >$ `= primals(`$a, optn$`)`		onedimensional homogeneity analysis of multimonial data
$< gof, eval, load, scor, catq, single, multi > =$ `= princals(`$data, scale, optn$`)`		principal component analysis of categorical data (Gifi, 1990)
$< d, rot, c >$ `= procrust(`$a, b <, par <, nrow >>$`)`		orthogonal Procrustes problem
$< plan, planb >$ `= promep(`$levs <, optn >$`)`		MinPROMEP: partially replicated minimal orthogonal main-effect plans
$quant$ `= quantile(`$a, k \vert prob <, optn >$`)`		obtain k quantile nine different types like in R function
$< gof, misc, pred, gini, prox, outl, intact >$ `=` `= rafprd(`$infor, data, modl <, optn <, class <, cwgt > . >$`)`		Random Forest prediction (scoring) (based on the model obtained by `ranfor()`)
$< gof, misc, pred, gini, prox, outl, inact, tmis, tprd, tgini >$ `=` `= ranfor(`$data, modl <, optn <, class <, cwgt <, test > . >$`)`		Random Forest modeling (Breiman) Regression and Classification
$< gof, beta, ase, conf, cov, res >$ `= reg(`$data, model <, sopt <, optn <, class >>>$`)`	sopt=	linear regression analysis
	"l_2"	least squares regression
	"l_1"	L_1 or LAV regression
	"l_inf"	L_∞ or Chebychev regression
	"l_p"	L_p regression $p \geq 1$
	"odis"	orthogonal distance regression
	"lms"	least median squares regression
	"lts"	least trimmed median regression
	"hub"	some Huber regression methods
$< gof, area, diff, cova, covd >$ `=` `= roccomp(`$yvec, xmat <, contr <, optn >>$`)`		comparison of areas under ROC curve
$nsamp$ `= sampallo(`$data, vubc <, optn <, cost >>$`)`		optimal sample allocation in strata
$samp$ `= sample(`$nsmp, nobs \vert prob <, optn >$`)`		equal or unequal probability sampling with or without replacement
$< ind, smp >$ `= sampmd(data<,optn>)`		maximum distance sampling from data set
$< gof, parm, gcvtab, cov, restrn, restst >$ `=` `= scad(`$meth, xtrn, lamb, frac, model, optn <, class <, xtst >>$`)`		Smoothly Clipped Absolute Deviations
	"meth"	method for variable selection (Fan and Li, 2001)
	"lse"	linear least squares regression
	"phr"	proportional hazards regression
	"svm"	support vector machines regression
$vest$ `= scalpha(`$data <, optn >$`)`		sample coefficient alpha (Cronbach, 1951)
$< nval, infl, lrgof, malin >$ `= screetst(`$eval, optn$`)`		scree test for eigen and singular values

Statistical Functions (Contd.)		
Function	**Spec**	**Short Description**
$< gof, klc, eklc, spm, spr, spc, wspm, wspr, wspc > =$ `= sdcspm(`$D <, rho <, optn >>$`)`		DOE search probability measures of Design matrices
$< nxtgen, nxtforb >$ `= selc(`$inides, forbid, nlev, optn$`)`		sequential elimination of level combinations
$< gof, est, resi, toteff, indeff >$ `= sem(`$data, model, optn <, parms <, wmat <, lc <, repar <, nlc, nlcb >>>>>$`)`		(mean and) covariance structure analysis and (linear) structural equations
$< gof, parm, shape, ase, cov >$ `= sgmanova(`$data, modl <, class <, optn >>$`)`		robust MANOVA based on spatial signs
$< parm, stder, conf, cov >$ `= simex(`$data, model, vardis <, optn <, class >>$`)`		simulation extrapolation
$< gof, parm, atst, ptst, res >$ `= sir(`$a, model <,$"$sopt$" $<, optn <, slic <, class >>>>$`)`	sopt=	sliced inverse regression
	"sir"	sliced inverse regression
	"save"	sliced average variance estimation
	"rphd"	principal Hessian direction (residuals)
	"yphd"	principal Hessian direction (response)
	"qphd"	quadratic principal Hessian direction
$< z, locscal >$ `= stand(`$x <, sopt <, optn <, ls >>>$`)`		(columnwise) standardization
	sopt=	(location,scale)
	mea(n)	(mean,1)
	med(ian)	(median,1)
	sum	(0,sum)
	euc(len)	(0,Euclidean length)
	ust(d)	(0,Uncorrected Std. Deviation)
	std	(mean,Standard Deviation)
	ran(ge)	(minimum,range)
	mid(range)	(midrange,range/2)
	max(abs)	(0,max absolute value)
	iqr	(median,interquartile range)
	mad	(median,abs. deviation from median)
	biw(c)	iweight 1-step M-estim,biweight A-estimate)
	hub(er)(c)	(Huber 1-step M-estimate,Huber A estimate)
	wav(e)(c)	(Wave 1-step M-estimate,Wave A estimate)
	agk(p)	(mean,AGK (ACECLUS) estimate)
	spa(cing)(p)	(mid-minimum-spacing,minimum spacing)
	lpm(p)	(Lp location,Lp scale (FASTCLUS))
	inp	(input,input)
	rev	(reverse use of input,reverse use of input)
$< ttest, mom, ftest >$ `= ttest(`$data1, data2 <, optn >$`)`		uni- and bivariate t test and Welch's test
$< pval, cint, kwal >$ `= wilcox(`$xdata, ydata <, optn >$`)`		Wilcoxon rank sum test (Mann-Whitney test) and Wilcoxon signed rank test

Statistical Functions (Contd.)		
Function	**Spec**	**Short Description**
$< c, ase, conf >$ = `univar`$(a <, sopt < optn >>)$	sopt=	univariate function
	"min"	minimum value
	"max"	maximum value
	"rng"	range
	"ari"	arithemetic mean
	"med"	median
	"ust"	uncorrected standard deviation
	"var"	variance
	"std"	standard deviation
	"mad"	median absolute deviation (MAD)
	"sma"	scaled median absolute deviation
	"s_n"	robust scale S_n (Rousseeuw)
	"q_n"	robust scale Q_n (Rousseeuw)
	"ske"	skewness
	"kur"	kurtosis
	"qu1"	first quartile
	"qu3"	third quartile
	"iqr"	interquartile range
	"qua"	all three quartiles
	"loo"	L_∞ (maxabs) norm
	"lpm"	L_p norm, $p \geq 1$
	"fsp"	fourth spread (Hoaglin, 1983)
	"biw"	Tukey's biweight location
	"bis"	Tukey's biweight scal
	"hub"	Huber's location (Goodall, 1983) for $k > 0$
	"hus"	Huber's scale (Iglewicz, 1983) for $k > 0$
	"wav"	Andrew's wave location (Goodall, 1983) for $c > 0$
	"was"	Andrew's wave scale (Iglewicz, 1983) for $c > 0$
	"msp"	Minimum spacing location (Sarle, 1995) for $0 < p < 1$.
	"mss"	Minimum spacing scale (Sarle, 1995) for $0 < p < 1$.
$< effrep, outrep, cordat >$ = `urd1out`$(X, y <, optn >)$		unreplicated factorial designs with single outlier
$< gof, est, conf >$ = = `xctlog`$(data, model, exct <, optn < class >>)$		exact logistic regression by MCMC method

6.2.8 Data Mining

Data Mining Functions		
Function	**Spec**	**Short Description**
$< gof, asso, nset >$ = `assoc(`$cust, data, <, optn <, supp >>$`)`		associations of items
$< weights, add1, add2 >$ = `cluster(`$x, sopt$ $<, optn <, scal >>$`)`	sopt= "Agnes" "Clara" "Diana" "Fanny" "Mona" "Pam"	cluster methods Agglomerative (Hierarchical) Nesting Clustering Large Applications Divisive (Hierarchical) Analysis Fuzzy Analysis Monothetic Analysis Partitioning Around Medoids
$< gof, ccc, sv >$ = `cuclcr(`$data, cstr$ $<, optn <, sv <, rsq >>>$`)`		cubic cluster criterion
$< xful, scal >$ = `impute(`$xmis, sopt < optn$ $<, class <, bounds >>>$`)`	sopt= "scalar" "randuni" "randnrm" "colmean" "mindist" "knearn" "linreg" "simpls" "krnpls"	imputation of missing values impute scalar constant uniform random in column range columnwise (ν, σ) normal random constant column mean rowwise minimum distance rowwise K nearest neighbor columnwise linear regression columnwise linear PLS (SIMPLS) columnwise linear PLS (Kernel PLS)
$< gof, bmat, ecor, caval, cqord, cstat >$ = = `ita(`$data, optn$`)` $< gof, bmat, eqor, diff, cqord, cstat >$ = = `ita(`$data, optn$`)`		classic item tree analysis induct. item tree analysis
$< gof, alpha, beta, wgts, resi >$ = = `mvsvm(`$xtrn, ytrn <, optn <, kfun >>$`)`		multivariate SVM
$< gof, parm, fit, tabs, stat, scor, tscor >$ = = `nlfit(`$train, modl <, optn <, class <, fun1 <, fun2$ $<, actf <, link <, test > .. >$`)`		nonlinear data mining choosing from a set of nonlinear functions
$< gof, scor, fit, tabs >$ = = `nlfitprd(`$data, parm, stat, modl <, optn <, class$ $<, actf <, link > .. >$`)`		scoring for `nlfit()` with new data set
$< gof, pc, eval >$ = = `nlkpca(`$x, optn <, class <, kfun >>$`)`		(nonlinear) Kernel PCA
$< alfa, sres, vres, yptr, yptt >$ = `nlkpls(`$trn,$		(nonlinear) Kernel PLS
$< gof, tree, cltrn, prtrn, cltst, prtst >$ = = `recupar(`$trn, modl, optn, ord <, nom <, flt <, test > . >$`)`		recursive partitioning (chaid) (similar to SAS `treedisc` macro)
$< gof, rules >$ = `rules(`$asso <, optn >$`)`		rules in associations of items
$< gof, osequ >$ = `sequ(`$cust, visit, data, asso <, optn <, supp >>$`)`		sequences of items
$< gof, beta, yprd, errt >$ = = `smp(`$data, tau, model <, optn, < class >>$`)`		stochastic matching pursuit and componentwise Gibbs sampler

Data Mining Functions		
Function	**Spec**	**Short Description**
$< gof, beta, resi >$ =		relevance vector machines
= `rvm`$(xtrn, ytrn <, optn <, kfun >>)$		(Tipping,2001, Herbrich, 2002)
$< gof, theta, cmat, bvec, yprd, ftrn, yptt, ftst >$ =		SM support vector machines:
= `smsvm`$(task, train, lamb, model, optn,$	"task"	multicategory classification
$class <, test, <, t0 <, x0 <, kfun >>>>)$	"msvm"	multicategory SVM by Lee and Wahba
	"ssvm"	structured multicategory SVM by Y. Lee
$< gof, scal, xwgt, ywgt, nass,$		Self Organizing Maps
$tmtrn, cwtrn, ptrn, mtrn, tmtst, cwtst, ptst, mtst >$ =		Kohonen Maps
= `som`$(xtrn, ytrn <, optn <, neus <, epos <, xtst <, ytst > . >)$		CP-ANN, SKN, XYF
$< gof, nodes, levmap, yptr, yptt >$ =		binary tree regression
= `split`$(trn, model, optn <, class <, tst >>)$		with binary response
$< alfa, sres, vres, yptr, yptt, plan, tunerr, tunzer >$ =		support vector machines:
= `svm`$(train, model <, optn <, class <, x0 <, kfun <, test > . >)$	"imet"	classification and regression
	"FQP"	full QP method
	"DQP"	decomposed QP method (shrinking)
	"LSVM"	Lagrangian SVM
	"ASVM"	Active SVM
	"PSVM"	Proximal SVM
	"SSVM"	Smooth (and Reduced) SVM
	"SMO"	Sequential Minimal Optimization
$< alfa, sres, vres, yptr, yptt, plan, tunerr, tunzer >$ =		SVM feature selection
= `svmfsm`$(train, model <, optn <, class <, x0 <, kfun <, test > . >)$		for SVM classification and regression
sym = `svmmat`$(data, model <, optn <, class >>)$	kind	computes SVM kernel matrix
	"line"	linear function
	"poly"	polynomial function
	"rbf"	Gaussian radial basis function ($[0, 1]$)
	"rbf2"	mod. Gaussian radial basis function ($[0, 1]$)
	"rbfcs"	mod. Gaussian radial basis function ($[0, 1]$)
	"erbf"	exponential radial basis function
	"tanh"	sigmoid function (same as "sigm")
	"sigm"	sigmoid function (same as "tanh")
	"four"	Fourier function ($[-\pi/2, \pi/2]$)
	"spli"	spline function ($[0, 1]$)
	"anov"	anova function
	"curv"	curvspline function ($[0, 1]$)
	"bspl"	Bspline function ($[0, 1]$)
	"anob"	anova spline function ($[0, 1]$)

Data Mining Functions		
Function	**Spec**	**Short Description**
$< yptr, vec >$ = svmpred($test, alfa, train, model <, optn <, class <, kfun >>>$)		computes SVM predicted values (score test data set)
$< alfa, sres, vres, yptr, yptt, plan >$ = svmstw($train, model <, optn <, class <, test >>>$)		SVM stepwise feature selection for SVM classification and regression
$< gof, est, tree, scor, struct, trace >$ = = varclus($data, optn <, ingrp >$)		clustering variables
$< gof, mod >$ = varsel($data, optn, yind, xind$)		multiple variable selection
$< gof, mod >$ = varsel($data, optn, model <, class >$)		multiple variable selection

6.2.9 Survival Methods for the Analysis of Censored Data

Survival Functions		
Function	**Spec**	**Short Description**
$< gof, curv, dase >$ = `survcurv(`$sopt, data, modl, optn <, clas >$`)`		(Adjusted) Survival Curves
	sopt	
	"adjaal"	adjusted Aalen's additive model
	"adjcox"	adjusted Cox's proportional hazards model
	"aalen"	Aaalen: based on Cox PH estimates
	"tsiatis"	Tsiatis: based on Cox PH estimates
	"breslow"	Breslow: based on Cox PH estimates
	"kapme"	Kaplan-Meier: based on Cox PH estimates
	"kalpr"	Kalbfleisch-Prentice: based on Cox PH estimates
	"green"	Greenwood: based on Cox PH estimates
	"exact"	Exact: based on Cox PH estimates
	"fleha"	Fleming-Harrington: based on Cox PH estimates
	"efron"	Efron: based on Cox PH estimates
	"kapme2"	Kaplan-Meier: not based on any model
	"fleha2"	Fleming-Harrington: not based on any model
	"fh2"	FH2: not based on any model
$< gof, res >$ = `survprd(`$parm, covm, sopt, data, modl, optn <, clas >$`)`		Survival Regression Prediction
	sopt	scoring of test data
$< gof, parm, cov, res, tres >$ = `survreg(`$sopt, data, modl, optn <, clas <, test >>$`)`		Survival Regression
	sopt	
	"aalen"	Aalen's additive model
	"phcox"	Cox's proportional hazards model
	"extrem"	regression with extreme distribution
	"logist"	regression with logistic distribution
	"gauss"	regression with Gaussian distribution
	"weibul"	regression with Weibull distribution
	"loglog"	regression with loglogistic distribution
	"lognor"	regression with lognormal distribution
	"expon"	regression with exponential distribution
		(this is Weibull with fixed scale=1)
	"rayle"	Rayleigh distribution
		(this is Weibull with fixed scale of .5)

6.2.10 Analysis of Micro Array Data

Micro Array Functions		
Function	**Spec**	**Short Description**
a `=` `generead(`$fpath, resp, sep <, optn <, scod >>$`)`		input of comma and other separated file input of special micro array data sets
$< gof, parms, dnew, mu >$ `=` `=` `affvsn(`$data, optn <, ref >$`)`		Variance Stabilizing Normalization
$< gof, dnew >$ `=` `affrma(`$data, optn <, ref >$`)`		RMA method for normalizing microarray data

6.2.11 Time Series

Time Series Functions		
Function	**Spec**	**Short Description**
$< gof, coef, resi, forc >$ = = `arima(`$data, pord <, optn >$`)`		AutoRegressive Integrated Moving Average (ARIMA) algorithm
$< gof, est, root, ase, cis, cov, sco >$ = = `arma(`$y <, x <, ar <, ma <, optn <, p0 > . >$`)`		autoregressive moving-average method
$< gof, yhat >$ = = `armafore(`$np, nh, est, y <, x <, ar <, ma <, optn > . >$`)`		forecasting step of the autoregressive moving-average method
$< coef, err >$ = `armcov(`$data, ord <, optn >$`)`		modified covariance method
$< gof, est, root, ase, nwcov, cov, resi >$ = = `arhet(`$y, lagbnd <, optn >$`)`		heterogeneous autoregressive method
$< prob, stat >$ = = `berkow(`$y <, optn <,$"$dist$" $<, ... >>$`)`		Berkowitz testing for distributions like `kstest()`
$< coef, err >$ = `burg(`$data, ord <, optn >$`)`		Burg's method for moving average coefficients
$< ccov, cmu >$ = `cndcov(`$data, zmu, isel, optn$`)`		conditional covariance matrix
b = `covlag(`a, k`)`		sequence of lagged cross product matrices
$< gof, est, ase, cis, cov, sco >$ = = `garch(`$meth, y, x, o, q, p <, optn < x0 >>$`)`		ARCH, GARCH, TARCH, AVARCH, ZARCH, APARCH, EGARCH, AGARCH, NAGARCH, IGARCH, FIGARCH
$< gof, cov, phi >$ = `mburg(`$x, lags <, optn <, cov >>$		modified Burg algorithm
xyp = `mempsd(`$coef, resi <, optn >$`)`		power spectrum of autoregressive filter
xyp = `pwelch(`$data <, optn >$`)`		power spectrum of a time series by Welch
`forec = tslocfor(`$xdat, order <, optn >$`)`		forecasting using local model
$< relerr, inderr >$ = = `tsloctst(`$xdat, order <, optn >$`)`		error estimation for local model
acv = `tsmeas(`$xdat,$"$alcdis$", $b <, optn >$`)`	"alcdis"	algorithmic complexity: partitions of equal distance
acv = `tsmeas(`$xdat,$"$alcpro$", $b <, optn >$`)`	"alcpro"	algorithmic complexity: partitions with same probability
$< mse, nmse, nrmse, cc >$ = = `tsmeas(`$xdat,$"$arfit$", $m <, optn >$`)`	"arfit"	statistical errors of fit at lead times
$< mse, nmse, nrmse, cc >$ = = `tsmeas(`$xdat,$"$arprd$", $m <, optn >$`)`	"arprd"	statistical errors of prediction at lead times
$< bic, cumbic >$ = = `tsmeas(`$xdat,$"$bicorr$", $tau <, optn >$`)`	"bicorr"	bicorrelation
$cordim$ = `tsmeas(`$xdat,$"$cordim$", $tau, m, s <, optn >$`)`	"cordim"	correlation dimension
$< cdim, csum, cent >$ = = `tsmeas(`$xdat,$"$cordi2$" $<, optn >$`)`	"cordi2"	correlation sum, entropy, and dimension
$corent$ = `tsmeas(`$xdat,$"$corent$", $tau, m, r <, optn >$`)`	"corent"	approximate correlation entropy
$corrad$ = `tsmeas(`$xdat,$"$corrad$", $tau, m, r <, optn >$`)`	"corrad"	radii of given correlation sums
$corsum$ = `tsmeas(`$xdat,$"$corsum$", $tau, m, r <, optn >$`)`	"corsum"	correlation sum
$detfl$ = `tsmeas(`$xdat,$"$detfl$" $<, optn >$`)`	"detfl"	detrended fluctuation analysis
$< stat, pval, cval, lags, resi >$ = `tsmeas(`$xdat,$"$difu l$" $<, optn >$`)`	"difu l"	(augmented) Dickey-Fuller testing (with automatic lag selection)

Time Series Functions (Contd.)		
Function	**Spec**	**Short Description**
$< stat, pval >$ = `tsmeas(`$xdat$,"$grang$", $tau <, optn >$`)`	"grang"	Granger causality testing for different types of inference
$< est, ase, r2s, cov, resi >$ = `tsmeas(`$xdat$,"$arvec$", $tau <, optn >$`)`	"arvec"	vector AR modeling (homo- and heteroskedastic)
$< est, ase, r2s, cov, resi >$ = `tsmeas(`$xdat$,"$iresp$", $tau <, optn >$`)`	"iresp"	impulse response modeling for specified value of lead
$< ebnd, psv, freq >$ = = `tsmeas(`$xdat$,"$eband$", $band <, optn >$`)`	"eband"	energy in frequency band
$< ym, ymn, tima, tmimi, dmima >$ = = `tsmeas(`$xdat$,"$exfea$", $f, nsmp <, optn >$`)`	"exfea"	local extreme values for a window
$< xmin, xmax >$ = `tsmeas(`$xdat$,"$extrem$" $<, optn >$`)`	"extrem"	maxima and/or minima of a component
$falsnn$ = `tsmeas(`$xdat$,"$falsnn$", $tau, m <, optn >$`)`	"falsnn"	percentage of false nearest neighbors
$fann$ = `tsmeas(`$xdat$,"$falsn2$" $<, optn >$`)`	"falsn2"	fraction of false nearest neighbors
$< fsle, ulen, npnt >$ = = `tsmeas(`$xdat$,"$fslexp$" $<, optn >$`)`	"fslexp"	finite size Lyapunov exponent
$< mob, comp >$ = `tsmeas(`$xdat$,"$hjoer$" $<, optn >$`)`	"hjoer"	Hjoerth parameters mobility and complexity
$hurst$ = `tsmeas(`$xdat$,"$hurst$" $<, optn >$`)`	"hurst"	Hurst exponent
$< cor, cum, dec, zer >$ = = `tsmeas(`$xdat$,"$kenda$", $tau <, optn >$`)`	"kenda"	Kendall autocorrelation
$< mse, nmse, nrmse, cc >$ = `tsmeas(`$xdat$,"$larfit$", $tau, m, nn, t <, optn >$`)`	"larfit"	in-sample direct predictions with a local model
$< mse, nmse, nrmse, cc >$ = = `tsmeas(`$xdat$,"$larprd$", $tau, m, nn, t <, optn >$`)`	"larprd"	direct predictions with a local model
$< stat, prob >$ = = `tsmeas(`$xdat$,"$ljung$" $<, optn >$`)`	"ljung"	Ljung-Box Test for serial correlation
$< stat, prob >$ = `tsmeas(`$xdat$,"$lmtst$" $<, optn >$`)`	"lmtst"	LM Test for serial correlation
$lyap$ = `tsmeas(`$xdat$,"$lyapk$" $<, optn >$`)`	"lyapk"	largest Lyapunov exponent (Kantz, 1994)
$lyap$ = `tsmeas(`$xdat$,"$lyapr$" $<, optn >$`)`	"lyapr"	largest Lyapunov exponent (Rosenstein et al., 1993)
$< medf, psv, freq >$ = = `tsmeas(`$xdat$,"$medfr$" $<, optn >$`)`	"medfr"	median frequency in range
$< mut, cummut, minmut >$ = = `tsmeas(`$xdat$,"$mutdis$", $tau, b <, optn >$`)`	"mutdis"	minimum of mutual information
$< mut, cummut, minmut >$ = = `tsmeas(`$xdat$,"$mutpro$", $tau, b <, optn >$`)`	"mutpro"	minimum of mutual information
$rmut$ = `tsmeas(`$xdat$,"$mutual$" $<, optn >$`)`	"mutual"	time delayed mutual information
$nwcov$ = `tsmeas(`$xdat$,"$nwcov$" $<, optn >$`)`	"nwcov"	Newey-West covariance matrix
$< gof, est, cov, yhat >$ = = `tsmeas(`$ydat$,"$nwreg$", $xdat <, optn >$`)`	"nwreg"	Newey-West regression with HAC standard errors

Time Series Functions (Contd.)		
Function	**Spec**	**Short Description**
$< pacor, ase >$ = = `tsmeas(`$xdat$,"$pacor$", $tau <, optn >$`)`	"pacor"	partial correlations with robust asymptotic standard errors
$< cor, cum, dec, zer >$ = = `tsmeas(`$xdat$,"$pears$", $tau <, optn >$`)`	"pears"	Pearson autocorrelation
$renyi$ = `tsmeas(`$xdat$,"$renent$" $<, optn >$`)`	"renent"	Renyi entropy of Qth order
$< cor, cum, dec, zer >$ = = `tsmeas(`$xdat$,"$spear$", $tau <, optn >$`)`	"spear"	Spearman autocorrelation
$corr$ = `tsmeas(`$xdat$,"$xcorr1$" $<, optn >$`)`	"xcorr1"	autocorrelation
$corr$ = `tsmeas(`$xdat$,"$xcorr2$" $<, optn >$`)`	"xcorr2"	cross correlations among time series data
$ydat$ = `tstrans(`$xdat$,"$sopt$" $<, optn <, add >>$`)`	"sopt"	transformations for time series data
	"aaft"	Amplitude Adjusted Fourier Transform Theiler et al (1992)
	"baki"	Baxter-King filtering
	"boxc"	Box-Cox transform depending on λ
	"four"	Fourier Transform surrogate
	"gaus"	marginal cumulative function to Gaussian
	"hopr"	Hodrick-Prescott filtering
	"iaaf"	Iterated Amplitude Adjusted Fourier Schreiber & Schmitz (1996)
	"line"	transform linearly to the interval $[0, 1]$
	"lagd"	lag difference transform depending on lag(s)
	"logd"	log difference transform depending on lag(s)
	"l121"	simple (iterated) 1-2-1 filter (see TISEAN)
	"norm"	transform to zero mean and unit standard deviation
	"notc"	Notch filter (see TISEAN)
	"perm"	random permuted data surrogate
	"sago"	Savitzky-Golay filter (see TISEAN)
	"stap"	statistically transformed AR process Kugiumtzis (2002b)
	"unif"	marginal cumulative function to Uniform in $[0, 1]$
$< gof, newdata >$ = = `x11(`$data <, optn <, inidate >>$`)`		seasonal adjustment with X11 algorithm

6.2.12 Probability and Combinatorics

Probability Functions		
Function	**Spec**	**Short Description**
p = `adprob(n,d,"vers")`		Probability of Anderson Darling CDF
p = `adtest(x,"vers")`		Anderson-Darling test (uniform and normal)
$< v, der >$ = `airy(`$z <$,"esc" $>$`)`		Airy A_i function
$< v, der >$ = `airybi(`$z <$,"esc" $>$`)`		Airy B_i function
$< prob, stat >$ = `berkow(`$a, optn$,"$dist$" $< ... >$`)`	dist	Berkowitz test for univariate distr. (for distributions see `kstest()`
$< y, der1, der2 >$ = `beta(`p, q`)`		Beta function
$< y, der1, der2 >$ = `betaln(`p, q`)`		log Beta function
$< y, der1, der2 >$ = `betainc(`p, q, x`)`		incomplete Beta function
y = `betamis(`p, x, a, b`)`		Beta function (m.v.)
y = `binomis(`$p, s, prob, n$`)`		Binomial function (m.v.)
p = `binorm(`h, k, r`)`		Bivariate Normal Distribution
v = `cdf23(`"$dist$",$lim, corr <, par >$`)`	"dist"	bi- and trivariate CDF functions
	"norm"	normal distribution
	"t"	t distribution
p = `cdf(`$dist, quant <, par_1, \ldots, par_k >$`)`	dist	cumulative density function
("bern",x, p)	"bern"	Bernoulli distribution
("beta",$x, a, b <, l <, u >>$)	"beta"	Beta distribution
("bin",$s, prob, n$)	"bin"	Binomial distribution
("cau",$x, <, \mu <, \sigma >>$)	"cau"	Cauchy distribution
("chis",$x, df <, nc >$)	"chis"	(noncentral) ChiSquare distribution
("expo",$x <, \sigma >$)	"expo"	Exponential distribution
("f",$x, ndf, ddf <, nc >$)	"f"	(noncentral) F distribution
("gam",$x, shape <, scale >$)	"gam"	Gamma distribution
("gaus",$x <, \mu <, \sigma >>$)	"gaus"	Gauss (Normal) distribution
("geom",m, p)	"geom"	Geometric distribution
("hypg",$x, m, k, n <, r >$)	"hypg"	Hypergeometric distribution
("igau",$x, shape$)	"igau"	Inverse Gauss (Wald) distribution
("lapl",$x <, \mu <, \sigma >>$)	"lapl"	Laplace distribution
("logi",$x <, \mu <, \sigma >>$)	"logi"	Logistic distribution
("logn",$x <, \mu <, \sigma >>$)	"logn"	LogNormal distribution
("negb",$x, prob, n$)	"negb"	Negative Binomial distribution
("norm",$x <, \mu <, \sigma >>$)	"norm"	Normal (Gauss) distribution
("pare",$x, a <, k >$)	"pare"	Pareto distribution
("pois",n, λ)	"pois"	Poisson distribution
("t",$t, df <, nc >$)	"t"	t distribution
("unif",$x <, l <, u >>$)	"unif"	Uniform distribution
("wald",x, d)	"wald"	Wald (Inverse Gauss) distribution
("weib",x,shape<,scale>)	"weib"	Weibull distribution
v = `cdfmv(`"$dist$",$ilim <, mu <, cov <, par >>>$`)`	"dist"	multivariate CDF functions
	"norm"	normal distribution
	"t"	t distribution

Probability Functions (Contd.)		
Function	**Spec**	**Short Description**
y = chimis($p, x, df <, nc >$)		ChiSquare function (m.v.)
c = combn($x <, n >$)		generates all combinations of x taken n at a time.
c = combn2(x)		generates all combinations of x taken $n = 2$ at a time
p = diehd($func <, optn >$)	test	test of uniform random generators
	"gcd"	greatest common denominator test
	"bda"	birthday spacings test
	"gor"	gorilla test (extensive "monkey" test)
r = dixonr($sopt, i, j, n, r\|R\|\alpha <, opt >$)		pdf, cdf, and critical values of Dixon's r
d = dmnom($x, prob$)		computes the density of multinomial distributed points
v = erf(x)		error function
v = erfcom($x <$,"esc" $>$)		complementary error function
v = erfinv(x)		inverse error function
$< p, dens, err >$ = fgen($y, nu, wgt <, par >$)		cdf of generalized F distribution
y = fmis($p, f, ndf, ddf <, nc >$)		F function (m.v.)
y = gammamis($p, x, shape, scale$)		Gamma function (m.v.)
$< y, der1, der2 >$ = gamma(x)		Gamma function
$< y, der1, der2 >$ = gammaln(x)		log Gamma function
$< y, der1, der2 >$ = gammainc(p, x)		incomplete Gamma function
z = hcube($x <, tran <, scal >>$)		generates all points on a hypercube lattice.
$quant$ =		inverse CDF (percent point function) for
= icdfmv($dist, prob, mu <, sigma <, par >>$)		multivariate normal and t distribution
$< prob, stat >$ =		Jarque-Bera test for univariate normal dist.
= jarbera($y <, optn >$)		(skewness and kurtosis)
p = ksprob($n, d <, sopt >$)	sopt	probability of Kolmogorov CDF
		and the complementary problem
	"bar"	compute complementary problem
	"sle"	Simard & L'Ecuyer algorithm
	"fap"	fast version of Simard & L'Ecuyer
	"mtw"	Marsaglia-Tsang-Wang algorithm

Probability Functions (Contd.)		
Function	**Spec**	**Short Description**
$< prob, stat >$ `=` `kstest(`$a, optn,$"$dist$" $< ... >$`)`	dist	Kolmogorov-Smirnov test for univariate distr.
	"bern"	Bernnoulli distribution
	"beta"	Beta distribution
	"bin"	Binomial distribution
	"cau"	Cauchy distribution
	"chis"	Chisquare distribution
	"expo"	Exponential distribution
	"f"	(noncentral) F distribution
	"gam"	Gamma distribution
	"gaus"	Gauss distribution
	"geom"	Geometric distribution
	"hypg"	Hypergeometric distribution
	"igau"	inverse Gauss distribution
	"lapl"	Laplace distribution
	"logi"	Logistic distribution
	"logn"	Lognormal distribution
	"negb"	negative Binomial distribution
	"nor"	normal distribution
	"par"	Pareto distribution
	"pois"	Poisson distribution
	"t"	(noncentral) t distribution
	"unif"	uniform distribution
$< conf, vol >$ `=` `mnprop(`$v <, alpha <, meth >>$`)`	meth	CIs for multinomial proportions
	"sison"	compute Sison-Glaz intervals
	"goodman"	compute Goodman intervals
`mrand(`$kind <, a, b >$`)`	kind=	creates multivariate random matrix
	"mnor"	multivariate normal $\mathcal{N}(\mu, \Sigma)$
	"mnom"	multinomial for scalar n and r vector p
	"unis"	uniformly distributed inside n dimensional sphere
	"unos"	uniformly distributed on n dimensional sphere
	"unie"	uniformly distributed inside n dimensional unit cube
	"unoe"	uniformly distributed on n dimensional unit cube
$< vec, contr >$ `=` `multst(`$options$`)`		multiple testing and simultaneous confidence intervals
vec `=` `mvelps(`$cov, mu, u, c, optn$`)`		multivariate normal and multivariate t probabilities over ellipsoidal regions noncentral F and χ^2 distribution
y `=` `nbinmis(`$p, s, prob, n$`)`		Negative Binomial function (m.v.)
n `=` `nsimplex(`p, n`)`		computes the number of points on a (p, n) simplex number p-part compositions of n
y `=` `normis(`p, x, μ, σ`)`		Normal function (m.v.)
T `=` `owenst(`h, a`)`		Owen's T function
$pout$ `=` `padjust(`$imet$, $pin <, par <, covm >>$`)`	imet=	multivariate p adjustment
	bon	Bonferroni
	hol	Holm
	hom	Hommel
	hoc	Hochberg
	bho	Benjamini & Hochberg
	bye	Benjamini & Yekutieli

Probability Functions (Contd.)		
dens = `pdf(`*dist*, *quant* < $par_1, \ldots, par_k$ >`)`	dist	probability density function
("bern",*x*, *p*)	"bern"	Bernoulli distribution
("beta",*x*, *a*, *b* <, *l* <, *u* >>)	"beta"	Beta distribution
("bin",*s*, *p*, *n*)	"bin"	Binomial distribution
("cau",*x* <, μ <, σ >>)	"cau"	Cauchy distribution
("chis",*x*, *df* <, *nc* >)	"chis"	(noncentral) ChiSquare distribution
("expo",*x* <, σ >)	"expo"	Exponential distribution
("f",*f*, *ndf*, *ddf* <, *nc* >)	"f"	(noncentral) F distribution
("gam",*x*, *shape* <, *scale* >)	"gam"	Gamma distribution
("gaus",*x* <, μ <, σ >>)	"gaus"	Gauss (Normal) distribution
("geom",*m*, *p*)	"geom"	Geometric distribution
("hypg",*x*, *m*, *k*, *n* <, *r* >)	"hypg"	Hypergeometric distribution
("igau",*x*, *shape*)	"igau"	Inverse Gauss (Wald) distribution
("lapl",x<, μ <, σ >>)	"lapl"	Laplace distribution
("logi",x<, μ <, σ >>)	"logi"	Logistic distribution
("logn",x<, μ <, σ >>)	"logn"	LogNormal distribution
("negb",*m*, *p*, *n*)	"negb"	Negative Binomial distribution
("norm",*x* <, μ <, σ >>)	"norm"	Normal (Gauss) distribution
("pare",*x*, *a* <, *k* >)	"pare"	Pareto distribution
("pois",*n*, λ)	"pois"	Poisson distribution
("t",*t*, *df* <, *nc* >)	"t"	t distribution
("unif",*x* <, < *l* >, < *u* >>)	"unif"	Uniform distribution
("wald",*x*, *d*)	"wald"	Wald (Inverse Gauss) distribution
("weib",*x*, *shape* <, *scale* >)	"weib"	Weibull distribution
v = `pdfmv(`"*dist*", *quant*, *mu*, *cov* <, *par* >`)`	"dist"	multivariate density functions
	"norm"	normal distribution
	"t"	t distribution
dens = `= pdfmv(`*dist*, *q* <, *mu* <, *sigma* <, *optn* >>>`)`		probability density function for multivariate normal and *t*
y = `poismis(`*p*, *s*, λ`)`		Poisson function (m.v.)
pts = `pppd(`*type*, *mu*, *sig*, *skew*, *kurt* <, *ipri* >`)`		percentage points of Pearson distribs.
`rand(`< *nr*, *nc*, *type*, ... > < *dist*|*rank*, ... >`)`	type=	creates random matrix
	'g'	rectangular `rand(`*nr*, *nc*,' *g*',"*dist*", ...`)`
	'd'	diagonal `rand(`*n*, *n*,' *d*',"*dist*", ...`)`
	'u'	upper triangular `rand(`*n*, *n*,' *u*',"*dist*", ...`)`
	'l'	lower triangular `rand(`*n*, *n*,' *l*',"*dist*", ...`)`
	's'	symmetric `rand(`*n*, *n*,' *s*',"*dist*", ...`)`
	'r'	rectangular `rand(`*nr*, *nc*,' *r*' <, *rank* >`)`
	'e'	symmetric `rand(`*n*, *n*,' *e*' <, *rank* <, *elo*, *ehi* >>`)`
	'o'	column orthogonal `rand(`*nr*, *nc*,' *o*'`)`
	dist=	see table below

Probability Functions (Contd.)		
p = `randisc(`$nsmp$,"bin",$n,p<,imet<,ipri>>$`)`		binary discrete random variate
p = `randisc(`$nsmp$,"hyp",$n,m,N<,imet<,ipri>>$`)`		hypergeometric discrete random variate
p = `randisc(`$nsmp$,"poi",$lambda<,imet<,ipri>>$`)`		Poisson discrete random variate
z = `rmult(`n,p`)`		multinomial random generator
$<prob,stat>$ = `shapwilk(`$y<,optn>$`)`		Shapiro-Wilks test for univariate normal and Shapiro-Francia test
$dist$ = `simdid(`$a,optn$`)`		obtain similarities or distances of discrete distribs.
y = `tmis(`$p,t,df<,pr>$`)`		T function (m.v.)
$<est,ci>$ = `xctbinom(`$x,n,p<,optn>$`)`		exact Binomial test
$p1$ = `xctbip1(`$powr,ssiz,p0<,optn>$`)`		$p(alt)$ of exact Binomial test
pow = `xctbipow(`$ssiz,p0,p1<,optn>$`)`		power of exact Binomial test
$size$ = `xctbissz(`$powr,p0,p1<,optn>$`)`		sample size for exact Binomial test
pow = `xctfipow(`$x,or<,optn>$`)`		power of exact Fisher test
$<est,ci>$ = `xctfishr(`$x,or<,optn>$`)`		exact Fisher test
$size$ = `xctfissz(`$x,or<,optn>$`)`		sample size for exact Fisher test
$pval$ = `xcthybr(`$tab<,optn>$`)`		hybrid exact Fisher test
$<est,ci>$ = `xctmcnem(`$x,or<,optn>$`)`		exact McNemar test
$<est,ci>$ = `xctpoiss(`$x,T,r<,optn>$`)`		exact Poisson test
$pval$ = `xctsimu(`$tab<,optn>$`)`		MC simu. of exact Fisher test
z = `xsimplex(`p,n`)`		generates all points of a (p,n) simplex number p-part compositions of n
$<r1,r2,r3,r4>$ = `zoverw(`$sopt,muz,sigz,muw,sigw,rho<,r<,opt>>$`)`		probability and density of z/w for normal z and w

6.2.13 Random Generators

Uniform Random Generators		
Distr.	**Add. Arg.**	**Description**
"icmp"	a, b	uniform RNG, very bad 16 bit version in Watcom C Compiler, int version
"iuni"	a, b	uniform with lower range a and upper range b, int version Mooore, RAND Corporation, see Fishman, p. 605
"iacm"	a, b	uniform random generator by Schrage (1979) in ACM, int version this is not a good choice
"ikis"	a, b	uniform random generator KISS by Marsaglia & Tsang, int version
"iecu"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), int version
"iec2"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), int version
"imwc"	a, b	multiply-with-carry RNG (Marsaglia, 2003), period 2^{128}, int version
"ix128"	a, b	XOR RNG (Marsaglia, 2003), period 2^{128}, int version
"iwow"	a, b	modified XOR RNG (Marsaglia, 2003), period $2^{192} - 2^{32}$, int version
"imet"	a, b	Mersenne-Twister (Matsumoto & Nishimura, 1998), int version
"iase"	a, b	uniform AES RNG (Hellakalek & Wegenkittel, 2003), int version
"igfs"	a, b	uniform GFSR4 RNG (Ziff, 1998), int version
"ilux"	a, b	uniform RANLUX RNG (Lüscher, 1994), int version
"itsh"	a, b	twin source hexadecimal (Richarson,2011), int version
"its7"	a, b	twin source base 256 (Richarson,2011), int version
"itss"	a, b	single source base 256 (Richarson,2011), int version
"its4"	a, b	twin source base 256 (Richarson,2011), int version
"dcmp"	a, b	uniform with lower range a and upper range b very bad 16 bit version in Watcom C Compiler, real version
"duni"	a, b	uniform with lower range a and upper range b Mooore, RAND Corporation, see Fishman, p. 605, real version
"dacm"	a, b	uniform random generator by Schrage (1979) in ACM, real version this is not a good choice
"dkis"	a, b	uniform random generator KISS by Marsaglia & Tsang, real version
"decu"	a, b	Tausworthe uniform random generator by L'Ecuyer (1996), real version
"dmwc"	a, b	multiply-with-carry RNG (Marsaglia, 2003), period 2^{128}, real version
"dx128"	a, b	XOR RNG (Marsaglia, 2003), period 2^{128}, real version
"dwow"	a, b	modified XOR RNG (Marsaglia, 2003), period $2^{192} - 2^{32}$, real version
"imet"	a, b	Mersenne-Twister (Matsumoto & Nishimura, 1998), real version
"dase"	a, b	uniform AES RNG (Hellakalek & Wegenkittel, 2003), real version
"dgfs"	a, b	uniform GFSR4 RNG (Ziff, 1998), real version
"dlux"	a, b	uniform RANLUX RNG (Lüscher, 1994), real version
"dtsh"	a, b	twin source hexadecimal (Richarson,2011), real version
"dts7"	a, b	twin source base 256 (Richarson,2011), real version
"dtss"	a, b	single source base 256 (Richarson,2011), real version
"dts4"	a, b	twin source base 256 (Richarson,2011), real version

Random Generator Distributions		
Distr.	**Add. Arg.**	**Description**
"beta"	α, β	Beta, $\mathcal{BE}(\alpha, \beta)$, with $\alpha > 0$ and $\beta > 0$; Randlib version (Brown et al. 1997, [118])
"bet2"	α, β	Beta, $\mathcal{BE}(\alpha, \beta)$, with $\alpha > 0$ and $\beta > 0$ version by Fishman (1996, [257])
"bino"	n, p	Binomial, $\mathcal{B}(n, p)$, with $n = 1, 2, \ldots$ and $0 < p < 1$ Randlib version (Brown et al. 1997, [118])
"bin2"	n, p	Binomial, $\mathcal{B}(n, p)$, with $n = 1, 2, \ldots$ and $0 < p < 1$ version by Fishman (1996, [257])
"cau1"	α, β	noncentral Cauchy, $\mathcal{C}(\alpha, \beta)$, with $-\infty < \alpha < \infty$ and $\beta > 0$ version by Fishman p. 192 (1996, [257])
"cau2"	α, β	noncentral Cauchy, $\mathcal{C}(\alpha, \beta)$, with $-\infty < \alpha < \infty$ and $\beta > 0$ version by Fishman p. 187 (1996, [257])
"chis"	df	chi square with $df > 0$ Randlib version (Brown et al. 1997, [118])
"exex"	λ	Double Exponential, $\mathcal{DE}(\lambda)$, version by Fishman p. 192 (1996, [257])
"expo"	λ	Exponential, $\mathcal{E}(\lambda)$, $\lambda > 0$ Randlib version (Brown et al. 1997, [118])
"exp2"	λ	Exponential, $\mathcal{E}(\lambda)$, $\lambda > 0$ *ziggurat* method by Marsaglia & Tsang (2000, [567])
"exp3"	λ	Exponential, $\mathcal{E}(\lambda)$, $\lambda > 0$ version by Fishman p. 192 (1996, [257])
"exp4"	λ	Exponential, $\mathcal{E}(\lambda)$, $\lambda > 0$ version by Fishman p. 189 (1996, [257])

Random Generator Distributions (Contd.)		
Distr.	**Add. Arg.**	**Description**
"frch"	α	Fréchet, $\mathcal{FR}(\alpha)$, with $\alpha > 0$ version by Zielinski(), p.106
"fsnd"	α, β	Snedecor's F, $\mathcal{F}(\alpha, \beta)$, with $\alpha > 0$ and $\beta > 0$ Randlib version (Brown et al. 1997, [118])
"fsn2"	α, β	Snedecor's F, $\mathcal{F}(\alpha, \beta)$, with $\alpha > 0$ and $\beta > 0$ version by Fishman p. 208 (1996, [257])
"gamm"	α, β	Gamma, $\mathcal{G}(\alpha, \beta)$, with $\alpha > 0$ and $\beta > 0$ Randlib version (Brown et al. 1997, [118])
"gam2"	α, β	Gamma, $\mathcal{G}(\alpha, \beta)$, with $\alpha > 0$ and $\beta > 0$ version by Fishman p. 193 (1996, [257])
"geom"	p	Geometric, $\mathcal{G}\rceil(p)$, with $0 < p < 1$
"hyge"	α, β, n	Hypergeometric, $\mathcal{H}(\alpha, \beta, n)$ with $\alpha > 0$, $\beta > 0$, and $1 \leq n \leq \alpha + \beta$ version by Fishman p. 218-220 (1996, [257])
"logn"	μ, σ	Lognormal, $\mathcal{LN}(\mu, \sigma)$, with mean μ and standard deviation $\sigma > 0$ version by Fishman (1996, [257])
"ncch"	df,$nonc$	noncentral chi square with $df > 0$ Randlib version (Brown et al. 1997, [118])
"ncfs"	α, β,$nonc$	noncentral F, $\mathcal{F}(\alpha, \beta, nonc)$, with $\alpha > 0$ and $\beta > 0$ Randlib version (Brown et al. 1997, [118])
"negb"	r, p	negative Binomial, $\mathcal{NB}(r, p)$, with $r > 0$ and $0 < p < 1$ Randlib version (Brown et al. 1997, [118])
"neg2"	r, p	negative Binomial, $\mathcal{NB}(r, p)$, with $r > 0$ and $0 < p < 1$ version by Fishman p. 222 (1996, [257])
"norm"	μ, σ	Normal, $\mathcal{N}(\mu, \sigma)$, with mean μ and standard deviation $\sigma > 0$ Randlib version (Brown et al. 1997, [118])
"nor2"	μ, σ	Normal, $\mathcal{N}(\mu, \sigma)$, with mean μ and standard deviation $\sigma > 0$ *ziggurat* method by Marsaglia & Tsang (2000, [567])
"nor3"	μ, σ	Normal, $\mathcal{N}(\mu, \sigma)$, with mean μ and standard deviation $\sigma > 0$ version by Fishman p. 190 (1996, [257])
"nor4"	μ, σ	Normal, $\mathcal{N}(\mu, \sigma)$, with mean μ and standard deviation $\sigma > 0$ version by Fishman p. 191 (1996, [257])
"pois"	λ	Poisson, $\mathcal{P}(\lambda)$, with $\lambda > 0$ Randlib version (Brown et al. 1997, [118])
"poi2"	λ	Poisson, $\mathcal{P}(\lambda)$, with $\lambda > 0$ version by Fishman p. 214 (1996, [257])
"poi3"	λ	Poisson, $\mathcal{P}(\lambda)$, with $\lambda > 0$

Random Generator Distributions (Contd.)		
Distr.	**Add. Arg.**	**Description**
"rayl"	σ	Rayleigh with $\sigma > 0$ (equivalent to Rice when $\mu = 0$) version by Zielinski(), p.106
"rice"	μ, σ	Rice with mean μ and $\sigma > 0$ version by Zielinski(), p.106
"tabl"	$[p_1, \ldots, p_n]$	tabled probability distribution with given table $0 \leq p_1 \leq p_2 \leq \ldots \leq p_n \leq 1$
"stud"	n	Student's t, $\mathcal{S}(n)$, with $n = 1, 2, \ldots$ version by Fishman p. 207 (1996, [257])
"tria"	h	Triangular distribution $0 < h < 1$
"univ"	μ, σ	uniform with mean μ and standard deviation $\sigma > 0$
"weib"	α, λ	Weibull, $\mathcal{W}(\alpha, \lambda)$, with $\alpha > 0$ and $\lambda > 0$

The keywords `"iuni", "iacm", "ikis", "iecu"` generate corresponding uniform integer random numbers in $[0, MACLONG]$.

6.2.14 Plotting

Plotting Functionality		
Function	**Spec**	**Short Description**
$< vol, box >$ = `boundbox(`xy`)`		minimum bounding box
$< gof, vert, offs, cent, neigh, norm >$ = `= convhull(`$x <, optn <, thresh <, bounds <, feapnt > . >$`)`		compute convex hulls (dim=2,3,...) (see `qhull` software)
$< gof, vert, offs, cent, neigh >$ = `= delaunay(`$x <, optn <, thresh <, bounds > . >$`)`		Delaunay triangulation (dim=2,3,...) (see `qhull` software)
rc = `gpbatch(`$gpfils$`)`		invoking gnuplot with input file(s)
`gnuplot ... gpend`		executing gnuplot interactively
`histplot(`$y <, titl <, optn <, fpath >>>$`)`		printer plotting of histograms
$< gof, yhat, ytst >$ = `loess(`$ytrn, xtrn,$ $optn <, wgt <, para <, drsq <, xtst >>>>$`)`		multivariate robust locally weighted regression (Cleveland)
$< xyp, res >$ = `lowess(`$xy <, optn >$`)`		(robust) locally weighted regression
$< vol, box >$ = `maxempty(`$xy <, xybc >$`)`		maximum empty box
$< gof, ap, xpa >$ = `propurs(`$data, nsol <, optn <, wgt >>$`)`		projection pursuit (Friedman and Tukey)
$< gof, vert, offs, cent, neigh >$ = `= voronoi(`$x <, optn <, thresh <, bounds > . >$`)`		Voronoi diagrams (dim=2,3,...) (see `qhull` software)
`xyplot(`$y, x <, titl <, optn <, labl <, fpath >>>>$`)`		printer plotting of (x, y) diagrams

6.2.15 Runtime Options

Runtime Options		
Option	**Spec**	**Short Description**
C_FIELDW=	*int*	field width of complex numbers with `print` statement
CENTER		center output of object with `print` statement
DEBUG=	*string*	only for debugging purposes
DECIMALS=	*int*	number of decimals real numbers with `print` statement.
ECHO		Echo input in .log output (default)
F_FIELDW=	*int*	field width of real numbers with `print` statement
I_FIELDW=	*int*	field width of integer numbers with `print` statement
INDBASE=	*int*	(also: IB) defines the lower index range (def. IB=1)
LINESIZE=	*int*	(also: LS) maximum number of chars online (def. LS=68)
NALLOC	*int*	set number of memory allocations for `lstmem` function
NAME		include name of the variable with `print` statement
NOCENTER		left flushed output with `print` statement (default)
NODEBUG		turning off the DEBUG option
NOECHO		suppress echo input in .log output
NONAME		opposite of NAME option (default)
NOPRINT		suppress all output to the .lst file
NOWSC		suppress work space compression (memory problem)
OPT_BS=	*int*	optimal block size used by LAPACK (def. OPT_BS=64)
PAGESIZE=	*int*	(also: PS) maximum number of lines printed on page (def. PS=60)
PRIME		return to prime version of subroutines (default)
PRINT		permits output to the .lst file after NOPRINT option (default)
RANDUNI		uniform RNG from RAND Corporation is used (SAS, IBM) (default)
RANDKISS		uniform RNG KISS by Marsaglia & Tsang (2002) (good)
RANDLECU		uniform RNG by L'Ecuyer (1999) is used (very good)
RANDLEC2		uniform RNG by L'Ecuyer (1999) is used (very good)
RANDMER		uniform Mersenne-Twister RNG (Matsumoto & Nishimura, 1998) (good)
RANDAES		uniform AES RNG (Hellakalek & Wegenkittel, 2003) (good)
RANDGFSR		uniform GFSR4 RNG (Ziff, 1998) (good)
RANDLUX		uniform RANLUX RNG (Lüscher, 1994) (good)
RANDXOR32		uniform RNG XOR32 is used, period $2^{32}-1$
RANDXOR64		uniform RNG XOR64 is used, 64bit, period $2^{64}-1$
RANDXORWOW		uniform RNG XORWOW is used, period $2^{192}-2^{32}$ (very good)
RANDXOR128		uniform RNG XOR128 is used, period $2^{128}-1$ (good)
RANDMWC3		uniform RNG MWC3 is used, period $2^{128}-1$ (good)
RANDCMP		uniform RNG of the host compiler is used (bad, period $2^{16}-1$)
RANDACM		uniform RNG by Schrage in ACM TOMS (1979), not good
RELZERO=	*real*	relative zero criterion
SECOND		run second version instead of prime version
SEED=	*int*	(re-) initialize the *seed* of `rand()`
SING=	*real*	criterion for singularity test (def. SING=1e-8)
SPRANGE=	*real*	sparsity range which (def. SPRANGE=.5)
SYMCRIT=	*real*	criterion for detection of symmetry (def. $\sqrt{meps}$)
USEUTF=	*int*	decide between work space or utility file (def. USEUTF=1000000)
WSC		(re-) permit (default) work space compression

Chapter 7

The Bibliography

Bibliography

[1] Abbasi, S. & Shaheen, F. (2008), "Faster generation of normal and exponential variates using the ziggurat method", Manuscript submitted to *JSS*.

[2] Abebe, A., Daniels, J., McKean, J.W., & Kapenga, J.A. (2001), *Statistics and Data Analysis*, `http://www.stat.wmich.edu/s160/book/`

[3] Abramowitz, M. & Stegun, I.A. (1972), *Handbook of Mathematical Functions*, Dover Publications, Inc., New York.

[4] Adlers, M. (1998), *Sparse Least Squares Problems with Box Constraints*, Linköping: Linköping University, Sweden.

[5] Agrawal, R., Imielski, T., & Swami, A (1993), "Mining association rules between sets of items in large databases"; Proceedings of the *ACM SIGMOD Conference on Management of Data*, p. 207-216.

[6] Agresti, A. (1996), *An Introduction to Categorical Data Analysis*, New York: John Wiley & Sons.

[7] Agresti, A. (2002), *Categorical Data Analysis*, Second Edition, New York: John Wiley & Sons.

[8] Ahn, H., Moon, H., Kim, S., & Kodell, R.L. (2002), "A Newton-based approach for attributing tumor lethality in animal carcinogenicity studies", *Computational Statistics and Data Analysis*, **38**, 263-283.

[9] Akcin, H. & Zhang, X. (2010), "A SAS Macro for direct adjusted survival curves based on Aalen's model"; *JSS*.

[10] Al-Baali, M. & Fletcher, R. (1985), "Variational Methods for Nonlinear Least Squares", *J. Oper. Res. Soc.*, **36**, 405-421.

[11] Al-Baali, M. & Fletcher, R. (1986), "An Efficient Line Search for Nonlinear Least Squares", *J. Optimiz. Theory Appl.*, **48**, 359-377.

[12] Al-Subaihi, A.A. (2002), "Variable Selection in Multivariable Regression Using SAS/IML", *JSS*, 2002.

[13] Amestoy, Davis, & Duff, I.S. (1996), "An approximate minimum degree ordering algorithm", *SIAM J. Matrix Analysis and Applic.* **17**, 886-905.

[14] Anderberg, M.R. (1973), *Cluster Analysis for Applications*, New York: Academic Press, Inc.

[15] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J. , Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., & Sorensen, D., (1995), *LAPACK User's Guide*, SIAM, Philadelphia, PA.

[16] Anderson, T. W. & Darling, D. A. (1954), "A test of goodness of fit", *Journal of the American Statistical Association*, **49**, 765-769.

[17] Andrei, N. (2007), "Scaled memoryless BFGS preconditioned conjugate gradient algorithm for unconstrained optimization"; *Optimization Methods and Software*, **22**, 561-571.

[18] Andrews, D.F., Bickel, P.J., Hampel, F.R., Huber, P.J., Rogers, W.H., Tukey, J.W. (1972), *Robust Estimation of Location: Survey and Advances*, Princeton NJ: Princeton University Press.

[19] Andrews, D.W.K. & Fair, R.C. (1988), "Inference in Nonlinear Econometric Models with Structural Change," *Review of Economic Studies*, **55**, 615-640.

[20] Anraku, K. (1999), "An Information Criterion for Parameters under a simple order restriction", *Biometrika*, **86**, 141-152.

[21] Appelgate, D., Bixby, R., Chvatal, V. & Cook, W.(2006), *Concorde TSP Solver*, `http://www.tsp.gatech.edu/concorde`.

[22] Appelgate, D., Bixby, R., Chvatal, V. & Cook, W.(2000), "TSP cuts which do not conform to the template paradigm" in M. Junger & D. Naddef (eds.): *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions*, *Lecture Notes in Computer Science*, Vol. 2241, pp. 261-304, London: Springer Verlag.

[23] Appelgate, D., Cook, W. & Rohe, A.(2003), "Chained Lin-Kernighan for large traveling salesman problems", *INFORMS Journal on Computing*, **15**, 82-92.

[24] Aranda-Ordaz, F.J. (1981), "On two families of transformations to additivity for binary response data," *Biometrika*, **68**, 357-364.

[25] Archer, C.O. & Jennrich, R.I. (1974), "Standard errors for rotated factor loadings"; *Psychometrika*, **38**, 581-592.

[26] Armstrong, R.D. & Kung, D.S. (1979), "Algorithm AS 135: Min-Max Estimates for a Linear Multiple Regression Problem", Appl. Statist. **28**, 93-100.

[27] Axelsson, O. (1996), *Iterative Solution Methods*, Cambridge University Press, Cambridge.

[28] Azzalini, A. & Capitanio, A. (1999), "Statistical applications of the multivariate skew-normal distribution", *Journal Roy. Statist. Soc. B* **61**, part 3.

[29] Azzalini, A. & Dalla Valle, A. (1996), "The multivariate skew-normal distribution", *Biometrika*, **83**, 715-726.

[30] Baker, F.B. (1992), *Item Response Theory: Parameter Estimation Techniques*, Marcel Dekker.

[31] Ballabio, D., Consonni, V., & Todeschini, R. (2009), "The Kohonen and CP-ANN toolbox: a collection of MATLAB modules of Self Organizing Maps and Counterpropagation Artificial Neural Networks"; *Chemometrics and Intelligent Laboratory Systems*, **98**, 115-122.

[32] Ballabio, D., Consonni, V., Vasighi, M., & Kompany-Zareh, M. (2011), "Genetic algorithm for architecture optimisation of counter-propagation artificial neural networks"; *Chemometrics and Intelligent Laboratory Systems*, **105**, 56-64.

[33] Ballabio, D. & Vasighi, M. (2011), "A MATLAB toolbox for self organizing maps and derived supervised neural network learning strategies"; *JSS*, 2011.

[34] Bamber, D. (1975), "The area above the ordinal dominance graph and the area below the receiver operating graph", *J. Math. Psych.*, **12**, 387-415.

[35] Bao, X. (2007), "Mining Transaction/Order Data using SAS Enterprise Miner Association Mode"; *SAS Global Forum*, Paper 132-2007.

[36] Barber, C.B., Dobkin, D.P., & Hubdanpaa, H.T. (1996), "The quickhull algorithm for convex hulls"; *ACM Trans. on Mathematical Software*.

[37] Barnett, V. & Lewis, T. (1978), *Outlier in Statistical Data*, New York: John Wiley & Sons.

[38] Barrodale, I. & Philips, F.D.K. (1975), "An improved algorithm for discrete Chebyshev linear approximation", in Hartnell, B.L., Williams, H.C (Eds.): *Proc. Fourth Manitoba Conf. on Numerical Mathematics*, Winnipeg 1975, 177-190.

[39] Barrodale, I. & Roberts, F.D.K. (1973), "An improved algorithm for discrete l_1 linear approximation", *SIAM Journal Numerical Analysis*, **10**, 839-848.

[40] Barrodale, I. & Roberts, F.D.K. (1974), "Algorithm 478: Solution of an overdetermined system of equations in the $l1$-norm",

[41] Bartels, R.E. & Stewart, G.W. (1972), "Solution of the equation AX + Xb = C", Comm. ACM, **15**, 820-26.

[42] Bartolucci, F. (2005), "Clustering univariate observations via mixtures of unimodal normal mixtures"; *Journal of Classification*, **22**, 203-219.

[43] Bates, D.M. & Watts, D.G. (1988), *Nonlinear Regression Analysis and Its Applications*, New York: John Wiley & Sons.

[44] Baxter, M. and King, R.G. (1999), "Measuring business cycles: Approximate band-pass filters for economic time series"; *The Review of Economics and Statistics*, **81**, 575-593.

[45] Beale, E.M.L. (1972), "A Derivation of Conjugate Gradients", in *Numerical Methods for Nonlinear Optimization*, F. A. Lootsma (ed.), London: Academic Press.

[46] Bellman, R. (1995), *Introduction to Matrix Analysis*, SIAM, Philadelphia, PA.

[47] Ben-Israel, A. & Greville, T.N. (1974), *Generalized Inverses: Theory and Applications*, New York: John Wiley & Sons.

[48] Benjamini, Y. & Hochberg, Y. (1995), "Controlling the false discovery rate: a practical and powerful approach to multiple testing", *Journal of the Royal Statistical Society B*, **57**, 289-300

[49] Benjamini, Y. & Liu, W. (1999), "A step-down multiple hypotheses testing procedure that controls the false discovery rate under interdependence", *Journal of Statistical Planning and Inference*, **82**, 163-170.

[50] Benjamini, Y. & Yekutieli, D. (2001), "The control of the false discovery rate in multiple testing under dependency", *Annals of Statistics*, **29**1165-1188.

[51] Benner, A., Ittrich, C., & Mansmann, U. (2006), "Predicting survival using microarray gene expression data," Technical Report, *German Cancer Research Institute*, Heidelberg, Germany.

[52] Benner, A. (2006), "Survival analysis in high dimensions," Technical Report, *German Cancer Research Institute*, Heidelberg, Germany.

[53] Bennett, K.P. (1999), "Combining support vector and mathematical programming methods for classification"; in: B. Schölkopf, C. Burges, & A. Smola (eds.): *Advances in Kernel Methods - Support Vector Machines*, pp. 307-326; Cambridge, MA: MIT Press.

[54] Bennett, K. P. & Embrechts, M. J.(2003), "An optimization perspective on kernel partial least squares regression", , .

[55] Bennett, K.P. & Mangasarian, O.L. (1992), "Robust linear programming discrimination of two linearly inseprable sets", *Optimization Software and Methods*, **1**, 23-34.

[56] Bentler, P.M. (1983), "Some Contributions to Efficient Statistics in Structural Models: Specification and Estimation of Moment Structures", *Psychometrika*, **48**,493-517.

[57] Bentler, P.M. (1989): *EQS, Structural Equations, Program Manual*, Program Version 3.0, Los Angeles: BMDP Statistical Software, Inc.

[58] Bentler, P.M. & Bonett, D.G. (1980), "Significance Tests and Goodness of Fit in the Analysis of Covariance Structures", *Psychological Bulletin*, **88**, 588-606.

[59] Bentler, P.M. & Weeks, D.G. (1980), "Linear Structural Equations with Latent Variables", *Psychometrika*, **45**, 289-308.

[60] Bentler, P.M. & Weeks, D.G. (1982), "Multivariate Analysis with Latent Variables", in *Handbook of Statistics, Vol. 2*, eds. P.R. Krishnaiah and L.N. Kanal, North Holland Publishing Company.

[61] Berkelaar, M., Eikland, K., Notebaert, P. (2004), *lp_solve (alternatively lpsolve)*, Open source (Mixed-Integer) Linear Programming system, Version 5.5.

[62] Berkowitz, J. (2001), "Testing density forecasts, with applications to risk management"; *Journal of Business and Economic Statistics*, **19**, 465-474.

[63] Bernstein, P. L. (1992), *Capital Ideas: The Improbable Origins of Modern Wall Street*, New York: The Free Press.

[64] Berry, M. B. & Browne, M. (1999), *Understanding Search Engines*, Philadelphia: SIAM.

[65] Berry, M. B. & Liang, M. (1992), "Large scale singular value computations", *International Journal of Supercomputer Applications*, **6**, pp. 13-49.

[66] Berry, M.J.A. & Linoff, G. (1997), *Data Mining for marketing, Sales, and Customer Support*, New York: J. Wiley and Sons, Inc.

[67] Bernaards, C.A. & Jennrich, R. A. (2004), "Gradient projection algorithms and software for arbitrary rotation criteria in factor analysis"; `http://www.stat.ucla.edu/research`; submitted for Publication.

[68] Betts, J. T. (1977), "An accelerated multiplier method for nonlinear programming", *Journal of Optimization Theory and Applications*, **21**, 137-174.

[69] Bi, J., Bennett, K.P., Embrechts, M., Breneman, C.M., & Song, M. (2002), "Dimensionality reduction via sparse support vector machines"; *Journal of Machine Learning*, **1**, 1-48.

[70] Billingsley, P. (1986), *Probability and Measure*, Second Edition, New York: J. Wiley.

[71] Birgin, E.G. & Martinez, J.M. (2001), "A spectral conjugate gradient method for unconstrained optimization"; *Appl. Math. Optim.*, **43**, 117-128.

[72] Bishop, Y.M., Fienberg, S. & Holland, P.W. (1975), *Discrete Multivariate Analysis*, Cambridge: MIT Press.

[73] Björck, A. (1996), *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, PA.

[74] Blaker, H. (2000), "Confidence curves and improved exact confidence intervals for discrete distributions"; *Canadian Journal of Statistics*, **28**, 783-798.

[75] Blanchard, G. & Roquain, E. (2008), "Two simple sufficient conditions for FDR control", *Electronic Journal of Statistics*, **2**, 963-992.

[76] Bock, R.D. (1972), "Estimating item parameters and latent ability when responses are scored in two or more nominal categories"; *Psychometrika*, **37**, 29-51.

[77] Bock, R.D. & Aitkin, M. (1981), "Marginal maximum likelihood estimation of item parameters: Application of an EM algorithm"; *Psychometrika*, **46**, 443-459.

[78] Boggs, P.T., Byrd, R.H., & Schnabel, R.B. (1987), "A stable and efficient algorithm for nonlinear orthogonal distance regression", *SIAM J. Sci. Stat. Comput.*, **8**, 1052-1078.

[79] Boggs, P.T., Byrd, R.H., Donaldson, J.R. & Schnabel, R.B. (1989), "Algorithm 676 - ODRPACK: Software for Weighted Orthogonal Distance Regression", *ACM TOMS*, **15**, 348-364.

[80] Boggs, P.T., Byrd, R.H., Rogers, J.E., & Schnabel, R.B. (1992), *Usersź Reference Guide for ODRPACK Version 2.01*, Technical Report NISTIR 92-4834, National Institute of Standards and Technology, Gaithersburg MD.

[81] Boik, R. J. & Robison-Cox, J.F. (1997), "Derivatives of the Incomplete Beta Function", paper submitted to *Journal of Statistical Software*.

[82] Bollen, K.A. (1986), "Sample Size and Bentler and Bonett's Nonormed Fit Index", *Psychometrika*, **51**, 375-377.

[83] Bollen, K.A. (1987), "Total, direct, and indirect effects in structural equation models", in: *Sociological Methodology*, C. C. Clogg (ed.), Washington, DC: American Sociological Association.

[84] Bollen, K.A. (1989 a), "A New Incremental Fit Index for General Structural Equation Models", *Sociological Methods and Research*, **17**, 303-316.

[85] Bollen, K.A. (1989 b), *Structural Equations with Latent Variables*, New York: John Wiley & Sons, Inc.

[86] Bollen, K.A. & Stine, R.A. (1990), "Direct and indirect effects: classical and bootstrap estimates of variablity", in: *Sociological Methodology*, C. Clogg (ed.), Washington, DC: American Sociological Association.

[87] Bollen, K.A. & Stine, R.A. (1992), "Bootstrapping goodness-of-fit measures in structural equation models", in: *Testing Structural Equation Models*, Bollen, K.A. & J.S. Long (eds.), Newbury Park: Sage.

[88] Bollerslev, T. (1986), "Generalized Autoregressive Conditional Heteroskedasticity," *Journal of Econometrics*, **31**, 307-327.

[89] Bolstad, B.M., Irizarry, R.A., Astrand, M, and Speed, T.P. (2003) "A Comparison of Normalization Methods for High Density Oligonucleotide Array Data Based on Bias and Variance"; *Bioinformatics*, **19**, 2, 185-193.

[90] Bonnett, D.G., Woodward, J.A., & Randall, R.L. (2002), "Estimating p-values for Mardia's coefficients of multivariate skewness and kurtosis", *Computational Statistics*, **17**, 117-122.

[91] de Boor, C. (1978), *A Practical Guide to Splines*, Berlin: Springer Verlag.

[92] Borchers, H. W. (2013), Package *adagio*, in CRAN.

[93] Borg, I. & Groenen, P. (2005), *Modern Multidimensional Scaling: Theory and Applications*; Berlin: Springer Verlag.

[94] Borg, I., Groenen, P.J.F., & Mair, P. (2010), "Multidimensionale Skalierung", Muenchen: Hampp Verlag.

[95] Botev, Z. I., Grotowski, J. F., and Kroese, D. P. (2010), "Kernel Density Estimation Via Diffusion", *Annals of Statistics*, **38**, 2916-2957.

[96] Bouaricha, A. & Moré, J.J. (1995), "Impact of Partial Separability on Large-scale Optimization", Technical Report MCS-P487-0195, Argonne National Laboratory, Argonne.

[97] Bowman, K.O. & Shenton, L.R. (1979), "Approximate percentage points of Pearson distributions"; *Biometrica*, **66**, 147-151.

[98] Bracken, J. & McCormick, G.P. (1968), *Selected Applications of Nonlinear Programming*, New York: John Wiley & Sons.

[99] Bradley, P.S. & Mangasarian, O.L. (1998), "Feature selection via concave minimization and support vector machines," in: J. Shavlik (ed.), *Machine Learning Proceedings of the Fifteenth International Conference*, 82-90, San Francisco: Morgan Kaufmann.

[100] Bradley, P.S. & Mangasarian, O.L. (1998), "Massive Data Discrimination via Linear Support Vector Machines", Technical Report 98-05, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[101] Bradley, P.S., Mangasarian, O.L., & Musicant, D.R. (1999), "Optimization methods in massive datasets," in: J. Abello, P.M. Pardalos, and M.G.C. Resende (eds), *Handbook of Massive Datasets*, Dordrecht, NL: Kluwer Academic Press.

[102] Brand, E.M. (2002), "Incremental singular value decomposition of uncertain data with missing values"; *European Conference on Computer Vision (ECCV)*, 2350 : 707-720.

[103] Breiman, L. (1993), "Better subset selection using the non-negative garotte"; *Technical Report*, Univ. of California, Berkeley.

[104] Breiman, L. (2001), "Random Forests", *Machine Learning*, **45**, 5-32.

[105] Breiman, L. & Cutler, A. (2001), "Random Forests", Technical Report `www.stat.berkeley.edu/ breiman`.

[106] Breiman, L. & Cutler, A. (2002), "How to use Survival Forests (SPDV1)", Technical Report `www.stat.berkeley.edu/ breiman`.

[107] Breiman, L., Friedman, J.H., Olshen, R.A., & Stone, C.H. (1984), *Classification and Regression Trees*, Wadsworth, Belmont CA.

[108] Brent, R. (1973), *Algorithms for Minimization Without Derivatives*, Prentice Hall, Inc.

[109] Breslow, N. E. & Day, N. E. (1980), *Statistical Methods of Cancer Research; Vol. I: The analysis of Case-Control Studies*, IARC Scientific Publications, IARC Lyon.

[110] Breslow, N. (2005), "Case-Control Study, Two-phase", in P. Armitage (ed.), *Encyclopedia of Biostatistics*, pp. 734-741, New York: J. Wiley & Sons.

[111] Bretz, F. (1999), *Powerful Modifications of William's Test on Trend*, Dissertation, Dept. of Bioinformatics, University of Hannover.

[112] Bretz, F., Hsu, J.C., Pinheiro, J.C. & Liu, Y. (2008), "Dose finding - a challenge in statistics", *Biometrical Journal*, **50**, 480-504.

[113] Bretz, F., Hothorn, T. & Westfall, P. (2011), *Multiple Comparisons Using R*, Boca Raton, London, New York: CRC Press.

[114] Bretz, F. & Hothorn, L.A. (1999), "Testing dose-response relationships with a priori unknown, possibly non-monotonic shapes", Technical Report, Dept. of Bioinformatics, University of Hannover.

[115] Brockwell, P.J., Dahlhaus, R., & Trindade, A.A. (2002), "Modified Burg algorithms for multivariate subset autoregression", Technical Report 2002-015, Dept. of Statistics, Univeristy of Florida.

[116] Bronstein, I.N. & Semendjajew, K.A. (1966), *Taschenbuch der Mathematik*, B.G. Teubner, Leipzig.

[117] Brown, B.W., Lovato, J., & Russell, K. (1997) *DCDFLIB: Library of Fortran Routines for Cumulatice Distribution Functions, Inverses, and Other Parameters*, Dept. of Biomathematics, University of Texas, Houston.

[118] Brown, B.W., Lovato, J., & Russell, K., Venier, J. (1997) *RANDLIB: Library of Fortran Routines for Random Number Generation*, Dept. of Biomathematics, University of Texas, Houston.

[119] Brown, R. G. (2009), "Dieharder: A random number test suite"; `www.phy.duke.edu/fgb/General/dieharder.php`

[120] Browne, M.W. (1974), "Generalized Least Squares Estimators in the Analysis of Covariance Structures", *South African Statistical Journal*, **8**, 1 - 24.

[121] Browne, M.W. (1982), "Covariance Structures", in *Topics in Multivariate Analyses*, ed. D.M. Hawkins, Cambridge University Press.

[122] Browne, M. W. (1984), "Asymptotically Distribution-free Methods for the Analysis of Covariance Structures", *Br. J. math. statist. Psychol.*, **37**, 62-83.

[123] Browne, M. W., (1992), "Circumplex Models for Correlation Matrices", Psychometrika, **57**, 469-497.

[124] Browne, M. W. (2001), "An overview of analytic rotation in exploratory factor analysis"; *Multivariate Behavioral Research*, **36**, 111-150.

[125] Browne, M. W. & Cudeck, R. (1993), "Alternative Ways of Assessing Model Fit", in: *Testing Structural Equation Models*, eds. K. A. Bollen & S. Long, Newbury Park: SAGE Publications.

[126] Browne, M. W. & Du Toit, S.H.C. (1992), "Automated Fitting of Nonstandard Models", *Multivariate Behavioral Research*, **27**, 269-300.

[127] Browne, M.W. & Shapiro, A. (1986), "The Asymptotic Covariance Matrix of Sample Correlation Coefficients under General Conditions", *Linear Algebra and its Applications*, **82**, 169-176.

[128] Brownlee, K.A. (1965), *Statistical Theory and Methodology in Science and Engineering*; New York: John Wiley & Sons.

[129] *BSD UNIX Programmer's Manual* (1989), Hewlett-Packard Comp., Palo Alto, CA.

[130] Bunch, J.R. & Kaufman, L. (1977), "Some stable methods for calculating inertia and solving symmetric linear systems", *Math. Comp.* **31**, 163-179.

[131] Burg, J. P. (1968), "A new analysis technique for time series data"; *NATO Advanced Study on Signal Processing with Emphasis on Underwater Acoustics*, Enschede, netherlands, August, 1968.

[132] Burnham, K. (1989), "Numerical Survival Rate Estimation for Capture-Recapture Models Using SAS PROC NLIN", in: L.McDonald, B. Manly, J. Lockwood, & J. Logan (Eds.): *Estimation and Analysis of Insect Populations*, Lecture Notes in Statistics **55**, Springer Verlag, Berlin-Heidelberg-New York.

[133] Bus, J.C.P. & Dekker, T.J. (1975), "Two efficient algorithms with guaranteed convergence for finding a zero of a function", *ACM TOMS*, **1**, 330-345.

[134] Butler, R.W., Davies, P.L., & Jhun, M. (1993), "Asymptotics for the Minimum Covariance Determinant Estimator", *The Annals of Statistics*, **21**, 1385-1400.

[135] Byrd, R.H., Lu, P., Nocedal, J., & Zhu, C. (1995), "A limited memory algorithm for bound constrained optimization", *SIAM Journal Scientific Computation*, **16**, 1190-1208.

[136] Byrd, R., Nocedal, J., & Schnabel,R. (1994) "Representations of Quasi-Newton Matrices and their use in Limited Memory Methods", *Mathematical Programming*, **63** no. 4, 129-156.

[137] Cai, L., Maydeu-Olivares, A., Coffman, D.L. & Thissen, D. (2006), "Limited information goodness of fit testing of item response theory models for sparse 2^p tables"; *British Journal of Mathematical and Statistical Psychology*, **59**, 173-194.

[138] Cain, K.C. & Breslow, N.E. (1988), "Logistic regression analysis and efficient design for two-stage studies", *American Journal of Epidemiology*, **128**, 1198-1206.

[139] Carbon, C.C. (2011), "BiDimRegression: A Matlab toolbox for calculating bidimensional regressions"; JSS, 2011.

[140] Carey, V., Zeger, S.L., & Diggle, K.Y., (1993) *Modelling multivariate binary data with alternating logistic regressions*, *Biometrika*, **80**, 517-526.

[141] Carlson, B.C. & Notis, E.M. (1979), "Algorithm 577: Algorithms for Incomplete Elliptic Integrals", *ACM TOMS*, **7**, 398-403.

[142] Carpaneto, G., Dell'Amico, M., & Toth, P. (1995), "A branch-and-bound algorithm for large scale asymmetric traveling salesman problems", Algorithm 750, *Transactions on Mathematical Software*, **21**, 410-415.

[143] Carroll, R., Gail, M., Lubin, J. (1993), "Case-control studies with errors in covariates", *Journal of the American Stat. Association*, **88**, 185-199.

[144] Carroll, R.A., Ruppert, D. & Stefanski, L.A. (1995), *Measurement error in Nonlinear Models*, London: Chapman and Hall.

[145] Carroll, R.A., Küchenhoff, H., Lombard, F., & Stefanski, L.A. (1996), "Asymptotics for the SIMEX estimator in nonlinear measurement error models", *JASA*, **91**, 242-25.

[146] Carroll, J.D. & Chang, J.J. (1970), "Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition"; *Psychometrika*, **35**, 283-320.

[147] Carroll, R., Gail, M., Lubin, J. (1993), "Case-control studies with errors in covariates", *Journal of the American Stat. Association*, **88**, 185-199.

[148] Catral, M., Han, L., Neumann, M. & Plemmons, R. J. (2004), "On reduced rank nonnegative matrix factorization for symmetric nonnegative matrices", *Linear Algebra and Applications*, **393**, 107-127.

[149] Cattell, R.B. (1966), "The scree test for the number of factors", *Multivariate Behavioral Research*, **1**, 245-276.

[150] Cattell, R.B. & Vogelmann, S. (1977), "A comprehensive trial of the scree and KG criteria for determining the number of factors", *Multivariate Behavioral Research*, **12**, 289-325.

[151] Chamberlain, R.M., Powell, M.J.D., Lemarechal, C., & Pedersen, H.C. (1982), "The watchdog technique for forcing convergence in algorithms for constrained optimization", *Mathematical Programming*, **16**, 1-17.

[152] Chan, T. F. (1982a), "An improved algorithm for computing the singular value decomposition", *ACM TOMS*, **8**, 72-83.

[153] Chan, T. F. (1982b), "Algorithm 581: An improved algorithm for computing the singular value decomposition", *ACM TOMS*, **8**, 84-88.

[154] Chan, T. F. (1987), "Rank revealing QR factorizations", *Linear Algebra and Its Applic.*, **88/89**, 67-82.

[155] Chang, C.M. (2003), "MinPROMEP: Generation of partially replicated minimal orthogonal main-effect plans using a novel algorithm", *JSS*, 2003.

[156] Chang, C.M. (2003), "Construction of partially replicated minimal orthogonal main-effect plans using a general procedure", *Utilitas Mathematica*, 2003.

[157] Chasalow, S. (2005), "The combinat Package"; Technical Report for R functions. See `CRAN` website.

[158] Chatterjee, S. & Price, B. (1977), *Regression Analysis by Example*, New York: John Wiley & Sons.

[159] Chauvenet, W. (1863), *A Manual of Spherical and Practical Astronomy*, Dover N.Y. 1960, 474-566.

[160] Chen, R.-B., Chu, C.-H., and Weng, J.-Z. (2010), "A stochastic matching pursuit (SMP) MATLAB toolbox for Bayesian variable selection"; *JSS*, 2010.

[161] Cheung, T. Y. (1980), "Multifacility location problem with rectilinear distance by the minimum cut approach", *ACM Trans. Math. Software*, **6**, 387-390.

[162] Christensen, R., Pearson, L.M., & Johnson, W. (1992), "Case deletion diagnostics for mixed models", *Technometrics*, **34**, 38-45.

[163] Chu, M. T. & Funderlik, R. E. (2002), "The centroid decomposition: Relationships between discrete variational decompositions and SVD's"; *SIAM Journal Matrix Anal. Appl.*, **23**, 1025-1044.

[164] Chung, J,K., Kannappan, P.L., Ng, C.T. & Sahoo, P.K. (1989), "Measure of distance between probability distributions"; *Journal of Mathem. Analysis and Applications*, **138**, 280-292.

[165] Clarke, G.P.Y. (1987), "Approximate Confidence Limits for a Parameter Function in Nonlinear Regression", *JASA*, **82**, 221-230.

[166] Clauß, G. & Ebner, H. (1970), *Grundlagen der Statistik für Psychologen, Pädagogen und Soziologen*, Frankfurt a.M. und Zürich, Verlag Harri Deutsch.

[167] Cleveland, W.S. (1979), "Robust locally weighted regression and smoothing scatterplots"; *JASA*, **74**, 829-836.

[168] Cleveland, W.S., Grosse, E. & Shyu, W.M. (1992), "Local regression models"; in: *Statistical Models in S*, eds. J.M. Chambers and T.J. Hastie; New York: Chapman & Hall.

[169] Cody, W.J. (1969), "Rational Chebyshev approximation for the error function", *Mathematics of Computation*, **23**, 631-637.

[170] Cody, W.J. (1993), "Algorithm 715", *ACM TOMS*, **19**, 22-32.

[171] Cody, W.J. & Waite, W. (1980), *Software Manual for the Elementary Functions*, Prentice Hall, Inc., Englewood Cliffs, NJ.

[172] Coleman, T.F., Garbow, B.S., & Moré, J.J. (1984), "Software for Estimating Sparse Jacobian Matrices", *ACM TOMS*, **10**, 329-345.

[173] Coleman, T.F., Garbow, B.S., & Moré, J.J. (1985), "Software for Estimating Sparse Jacobian Matrices", *ACM TOMS*, **11**, 363-377.

[174] Conn, A. R. & Gould, N.I.M (1984), "On the location of directions of finite descent for nonlinear programming algorithms", *SIAM Journal Numerical Analysis*, **21**, 1162-1179.

[175] Cook, R.D. (1998), *Regression Graphics*, New York: Wiley & Sons.

[176] Cook, J. R. & Stefanski, L.A. (1994), "Simulation Extrapolation in Parametric Measurement Error Models", JASA **89**, 1314-1328.

[177] Cook, R.D. & Weisberg, S. (1990), "Confidence curves for nonlinear regression", *JASA*, **85**, 544-551.

[178] Cook, R.D & Weisberg, S. (1994), *An Introduction to Regression Graphics*, New York: John Wiley & Sons.

[179] Cook, R.D. & Weisberg, S. (1999), *Applied Regression Including Computing and Graphics*, New York: Wiley. & Sons

[180] Copenhaver, T.W. and Mielke, P.W. (1977), "Quantit analysis: a quantal assay refinement," *Biometrics*, **33**, 175-187.

[181] Cormen, T.H., Leiserson, C.E., & Rivest, R.L. (1997), *Introduction to Algorithms*, Cambridge: MIT Press.

[182] Cornuejols, G. & Tütüncü, R. (2006), *Optimization Methods in Finance*, Pittsburgh PA: Carnegie Mellon University.

[183] Cox, C. (1998), "Delta Method", *Encyclopedia of Biostatistics*, eds. Armitage, P. & Colton, T., 1125-1127, New York: J. Wiley.

[184] Cox, D.R. & Hinkley, D.V. (1974), *Theoretical Statistics*, London: Chapman and Hall.

[185] Chamberlain, R.M.; Powell, M.J.D.; Lemarechal, C.; & Pedersen, H.C. (1982), "The watchdog technique for forcing convergence in algorithms for constrained optimization", *Mathematical Programming*, **16**, 1-17.

[186] Cooley, W.W. & Lohnes, P.R. (1971), *Multivariate Data Analysis*, New York: John Wiley & Sons, Inc.

[187] Cramer, J.S. (1986), *Econometric Applications of Maximum Likelihood Methods*, Cambridge: Cambridge University Press.

[188] Cristianni, N. & Shawe-Taylor, J. (2000), *Support Vector Machines and other kernel-based learning methods*, Cambridge University Press, Cambridge.

[189] Croes, G. A. (1958), "A method for solving traveling-salesman problems", *Operations Research*, **6**, 791-812.

[190] Cronbach, L.J. (1951), "Coefficient alpha and the internal structure of tests"; *Psychometrika*, **16**, 297-334.

[191] Croux, C., Filzmoser, P., Pison, G. & Rousseeuw (2004), "Fitting multiplicative models by robust alternating regressions"; Technical Report.

[192] Cudeck, R. & Browne, M.W. (1984), "Cross-validation of covariance structures", *Multivariate Behavioral Research*, **18** 62-83.

[193] Czyzyk, J., Mehrotra, S., Wagner, M. & Wright, S.J. (1997) "PCx User Guide (Version 1.1)", Technical Report OTC 96/01, Office of Computational and Technology Research, US Dept. of Energy.

[194] Dale, J.C. (1985), "A Bivariate Discrete Model of Changing Colour in Blackbirds", in *Statistics in Ornithology*, Morgan, B.J.T. and North, P.M. (eds.) Berlin: Springer-Verlag.

[195] Dale, J.C. (1986), "Global Cross-Ratio Models for Bivariate, Discrete, Ordered Responses", *Biometrics*, **42**, 909-917.

[196] Dalenius, T. (1957), *Sampling in Sweden. Contributions to Methods and Theories of Sample Survey Practice*; Stockholm: Almqvist & Wiksells.

[197] Daniel, J.W., Gragg, W.B. , Kaufman, L. & Stewart, G.W. (1976), "Reorthogonalization and Stable Algorithms for Updating the Gram-Schmidt QR Factorization", Math. Comp. **30**, 772-795.

[198] Davies, L. (1992), "The Asymptotics of Rousseeuw's Minimum Volume Ellipsoid Estimator", *The Annals of Statistics*, **20**, 1828-1843.

[199] Davison, A.C. & Hinkley, D.V. (1997), *Bootstrap Methods and their Application*, Cambridge: Cambridge University Press.

[200] Davison, A.C., Hinkley, D.V., & Schechtman, E. (1986), " Efficient bootstrap simulation", *Biometrika*, **73**, 555-566.

[201] Dayal, B.S. & McGregor, J.F. (1997), "Improved PLS Algorithms", *Journal of Chemometrics*, **11**, 73-85.

[202] Deak, I. (1980), "Three Digit Accurate Multiple Normal Probabilities", *Numer. Math.*, **35**, 369-380.

[203] De Berg, M., Cheong, O., & van Kreveld, M. (2008). *Computational Geometry: Algorithms and Applications*, New York: Springer, 2008.

[204] DeJong, S. (1993), "SIMPLS: An alternative approach to partial least squares regression", *Chemometrics and Intelligent Laboratory Systems*, **18**, 251-263.

[205] DeJong, S. & ter Braak, C.J.F. (1994), *Journal of Chemometrics*, **8**, 169-174.

[206] de Leeuw, J. (1977), "Applications of convex analysis to multidimensional scaling"; in: Barra, Brodeau, Romier, & van Cutsem (eds.): *Recent Developments in Statistics*, Amsterdam: North Holland Publishing Company.

[207] de Leeuw, J. (1983), "Models and Methods for the Analysis of Correlation Coefficients", *Journal of Econometrics*, **22**, 113-137.

[208] de Leeuw, J. (1984), "Canonical Analysis of Categorical Data"; Leiden: DSWO Press.

[209] de Leeuw, J. (1994), "Block relaxation methods in statistics"; in Bock, Lenski, & Richter (ed.): *Information Systems and Data Analysis*, Berlin: Springer Verlag.

[210] de Leeuw, J. (2007), see the following website for a set of R programs: `http://gifi.stat.ucla.edu/psychoR/`

[211] de Leeuw, J. & Heiser, W. (1977), "Convergence of correction-matrix algorithms for multidimensional scaling"; in J. C. Lingoes (ed.): *Geometric Representations of Relational Data*, Ann Arbor, MI: Mathesis Press.

[212] de Leeuw, J. & Pruzansky, S. (1978), "A new computational method to fit the weighted Euclidean distance model"; *Psychometrika*, **43**, 479-490.

[213] de Leeuw, J., Young, F. W. & Takane, Y. (1976), "Additive structure in qualitative data: An alternating least squares method with optimal scaling features"; *Psychometrika*, **41**, 471-503.

[214] DeLong E.R., DeLong D.M., & Clarke-Pearson, D.L. (1988), "Comparing the areas under two or more correlated receiver operating characteristic curves: A nonparametric approach," *Biometrics*, **44**, pp. 837-845.

[215] Dennis, J.E., Gay, D.M., & Welsch, R.E. (1981), "An Adaptive Nonlinear Least-Squares Algorithm", *ACM Trans. Math. Software*, **7**, 348-368.

[216] Dennis, J.E. & Mei, H.H.W. (1979), "Two new unconstrained optimization algorithms which use function and gradient values", *J. Optim. Theory Appl.*, **28**, 453-482.

[217] Dennis, J.E. & Schnabel, R.B. (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, New Jersey: Prentice-Hall.

[218] Diggle, P.J., Liang, K.Y., & Zeger, S.L. (1994) *Analysis of Longitudinal Data*, Oxford: Clarendon Clarendon Press, 1994.

[219] Ding, C., He, X., & Simon, H. D. (2005), "On the equivalence of nonnegative matrix factorization and spectral clustering", *Proc. SIAM Data Mining Conf.*, 2005.

[220] Ding, C., Peng, T.L., & Park, H. (2006), "Orthogonal nonnegative matrix tri-factorizations for clustering", .

[221] Dixon, W. J. (1950), "Analysis of extreme values"; *Annals of Mathematical Statistics*, **21**, 488-506.

[222] Dixon, W. J. (1951), "Ratios involving extreme values"; *Annals of Mathematical Statistics*, **22**, 68-78.

[223] *DOMAIN C Library (CLIB) Reference* (1985), Apollo Computer Inc., Chelmsford, MA.

[224] Dongarra, J.J., Bunch, J.R., Moler, C.B., & Stewart, G.W. (1979), *LINPACK User's Guide*, SIAM, Philadelphia, PA.

[225] Doornik, J. A. & Hansen, H. (2008): "An omnibus test for univariate and multivariate normality", *Oxford Bulletin of Economics and Statistics*, **70**, 927-939.

[226] Draper, N.R. & Smith, H. (1981), *Applied Regression Analysis*, New York: J. Wiley.

[227] Du, D.-Z. & Pardalos, P. M. (1993), *Network Optimization Problems - Algorithms, Applications, and Complexity*, Singapore: World Scientific.

[228] Duff, I.S. & Reid, J.K. (1978), *An implementation of Tarjan's algorithm for the block triangularization of a matrix*, ACM TOMS, **4**, 137-147.

[229] Duncan, G.T. (1978), "An Empirical Study of jackknife-constructed confidence regions in nonlinear regression", *Technometrics*, **20**, 123-129.

[230] Dunnett, C. W. (1980), "Pairwise multiple comparisons in the unequal variance case", *Journal of the American Statistical Association*, **75**, 796-800.

[231] Ecker, J. G. & Kupferschid, M. (1988), *Introduction to Operations Research*, (Reprint of 1991), Malabar, FL: Krieger Publishing Company

[232] Eckert, R. E. (1994), Purdue University, W. Lafayette, Personal Communication.

[233] Edlund, O. (1999), *Solution of Linear Programming and Non-Linear Regression Problems Using Linear M-Estimation Methods*, LuleåUniversity of Technology, Sweden.

[234] Efron, B. (1994), *The Jackknife, the Bootstrap, and Other Resampling Methods*, Philadelphia: SIAM.

[235] Efron, B. & Tibshirani, R.J. (1993), *An Introduction to the Bootstrap*, New York: Chapman & Hall.

[236] Efron, B., Hastie, T., Johnstone, I. & Tibshirani, R. (2002), "Least Angle Regression", *The Annals of Statistics*, **32**, 407-499.

[237] Einarsson, H. (1998), "Algorithms for General non-differentiable Optimization Problems"; Master of Science Thesis, IMM, Technical University of Denmark.

[238] Einarsson, H. & Madsen, K. (1999), "Sequential linear programming for non-differentiable optimization"; Paper presented at the *SIAM Meeting on Optimization*, Atlanta 1999.

[239] Elton, E. and Gruber, M. (1981), *Modern Portfolio Theory and Investment Analysis*, New York: John Wiley & Sons, Inc.

[240] Emerson, P.L. (1968), "Numerical construction of orthogonal polynomials from a general recurrence formula", *Biometrics*, **24**, 695-701.

[241] Engle, R.F. (1982), "Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of UK Inflation", *Econometrica*, **50**, 987-1008.

[242] Eskow, E. & Schnabel, R.B. (1991), "Algorithm 695: Software for a New Modified Cholesky Factorization", *ACM Trans. Math. Software*, **17**, 306-312.

[243] Esmailzadeh, N. (2013), "Two level search designs in Matlab", paper and software submitted to JSS.

[244] Everitt, B.S. (1984), *An Introduction to Latent Variable Methods*, London: Chapman & Hall.

[245] Everitt, B.S. (1996), *A Handbook of Statistical Analyses Using SAS*, London: Chapman & Hall.

[246] Fan, J., & Li, R. (2001), "Variable selection via nonconcave penalized likelihood and its oracle properties," *JASA*, **96**, pp. 1348-1360.

[247] Fan, J., & Li, R. (2002), "Variable selection for Cox's proportional hazards model and frailty model," *The Annals of Statistics*, **30**, pp. 74-99.

[248] Fay, M. P. (2010), "Two-sided exact tests and matching confidence intervals for discrete data", *R Journal*, **2**, 53-58.

[249] Ferguson, T. (1996), *A Course in Large Sample Theory*, London: Chapman & Hall.

[250] Ferson, W.E. and Harvey, C.R. (1992), "Seasonality and Consumption-Based Asset Pricing," *Journal of Finance*, **47**, 511-552.

[251] Fig, M. (2010), Matlab toolbox for permutations and combinations.

[252] Finney, D.J. (1947), "The estimation from individual records of the relationship between dose and quantal response", *Biometrika*, **34**, 320-334.

[253] Fisher, R. A. (1935), "The logic of inductive inference", *Journal of the Royal Statistical Society, Series A*, 39-54.

[254] Fisher, R.A. (1936), "The use of multiple measurements in taxonomic problems", *Annals of Eugenics*, **7**, part II, pp. 179-188.

[255] Fisher, R. A. (1962), "Confidence limits for a cross-product ratio", *Australian Journal of Statistics*, **4**, 41.

[256] Fisher, R. A. (1970), *Statistical Methods for Research Workers*, Oliver & Boyd.

[257] Fishman, G.S. (1996), *Monte Carlo: Concepts, Algorithms, and Applications*, New York: Springer Verlag.

[258] Fishman, G.S. & Moore, L.R. (1986), "An exhaustive analysis of multiplicative congruential random number generators with modulus 2**31-1", *SIAM Journal on Scientific and Statistical Computation*, **7**, p.24-45.

[259] Fletcher, R. (1987), *Practical Methods of Optimization*, 2nd Ed., Chichester: John Wiley & Sons.

[260] Fletcher, R. & Powell, M.J.D. (1963), "A Rapidly Convergent Descent Method for Minimization", *Computer Journal*, **6**, 163-168.

[261] Fletcher, R. & Xu, C. (1987), "Hybrid Methods for Nonlinear Least Squares", *Journal of Numerical Analysis*, **7**, 371-389.

[262] Floudas, C.A. & Pardalos, P.M. (1990), *A Collection of Test Problems for Constrained Global Optimization Algorithms*; *Lecture Notes in Computer Science*, **455**, Berlin: Springer Verlag.

[263] Forrest, J. J. & Lougee-Helmer, R. (2014), *Cbc*, `jjforre@us.ibm.com`, `robinlh@us.ibm.com`.

[264] Forrest, J. J., de la Nuez, D., & Lougee-Helmer, R. (2014), *Clp*, `http://www.tsp.gatech.edu/concorde`.

[265] Forsythe, G.E., Malcolm, M.A., & Moler, C.B. (1977), *Computer Methods for Mathematical Computations*, Englewood-Cliffs, NJ: Prentice Hall, 1977.

[266] Foster, D.P. & Stine, R.A. (2004), "Variable selection in Data Mining: Building a predictive model for bankruptcy"; JASA, **99**, 303-313.

[267] Fox, T., Hinkley, D. & Larntz, K. (1980), "Jackknifing in Nonlinear Regression", *Technometrics*, **22**, 29-33.

[268] Frank, I. & Friedman, J. (1993), "A statistical view of some chemometrics regression tools"; *Technometrics*, **35**, 109-148.

[269] Fraser, C. (1980), *COSAN User's Guide*, Toronto: The Ontario Institute for Studies in Education.

[270] Fraser, C. & McDonald, R.P. (1988), "NOHARM: Least squares item factor analysis", *Multivariate Behavioral Research*, **23**, 267-269.

[271] Fredman, M.L., Johnson, D.S., McGeoch, L.A. & Ostheimer, G. (1995), "Data structures for Traveling Salesmen", *J. Algorithms*, **18**, 432-479.

[272] Friedman, A., & Kohler, B. (2003), "Bidimensional regression: Assessing the configural similarity and accuracy of cognitive maps and other two-dimensional data sets", *Psychological Methods*, **8**, 468-491.

[273] Friedman, J.H., Bentley, J.L., Finkel, R.A. (1977), "An algorithm for finding best matches in logarithmic expected time", em ACM Transactions Math. Softw., p. 209-226, `http://doi.acm.org/10.1145/355744.355745`

[274] Friedman, J.H., & Tukey, J.W. (1974), "A projection pursuit algorithm for exploratory data analysis", J. Amer. Stat. Assoc. **62**, 1159-1178.

[275] Friedman, J. H. & Stuetzle, W. (1981), "Projection pursuit regression", *JASA*, **76**, 817-823.

[276] Fritsch, F.N. & Carlson, R.E. (1980), "Monotone piecewise cubic interpolation", *SIAM Journal Numerical Analysis*, **17**, 238-246.

[277] Froehlich, H., & Zell, A., (2004), "Feature subset selection for support vector machines by incremental regularized risk minimization", in: *The International Joint Conference on Neural Networks (IJCNN)*, **3**, 2041-2046.

[278] Fuller, W.A. (1987), *Measurement Error Models*, New York: J. Wiley & Sons.

[279] Fung, G. & Mangasarian, O.L. (2000), "Proximal Support Vector Machines", Technical Report, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[280] Fung, G. & Mangasarian, O.L. (2002), "Finite Newton method for Lagrangian support vector machine classification"; Technical Report 02-01, Data Mining Institute, Computer Sciences Dep., Univ. of Wisconsin, Madison, Wisconsin, 2002.

[281] Fung, G. & Mangasarian, O.L. (2003), "A Feature Selection Newton Method for Support Vector Machine Classification", *Computational Optimization and Applications*, 1-18.

[282] Gaevert, H., Hurri, J., Saerelae, J. & Hyvaerinen, A. (2005) see `http://www.cis.hut.fi/projects/ica/fastica/` for version 5

[283] Gallant, A. R. (1987), *Nonlinear Statistical Models*, New York: John Wiley & Sons, Inc.

[284] Garbow, B.S., Boyle, J.M., Dongarra, J.J., & Moler, C.B. (1977), *Matrix Eigensystem Routines - EISPACK Guide Extension*, *Lecture Notes in Computer Science*, vol. 51, Springer Verlag, Berlin.

[285] Gay, D.M. (1983), "Subroutines for Unconstrained Minimization", *ACM Trans. Math. Software*, **9**, 503-524.

[286] Genton, M. (2001), "Classes of kernels for machine learning: a statistics perspective", *Journal of Machine Learning Research*, **2**, 299-312.

[287] Gao, X., et al. (2008), "Nonparametric multiple comparison procedures for unbalanced one-way factorial designs", *Journal of Planning and Inference*, **77**, 2574-2591.

[288] Genz, A. (1986), "Fully Symmetric Interpolatory Rules for multiple Integrals", *SIAM Journal Numer. Analysis*, **23**, 1273-1283.

[289] Genz, A. (1991), *Computing in the 90s*, *Lecture Notes in Computer Science*, vol. 507, New York: Springer Verlag.

[290] Genz, A. (1992), "Numerical Computation of Multivariate Normal Probabilities", *J. of Computational and Graphical Stat.*, **1**, 141-149.

[291] Genz, A. (1999), "Numerical computation of critical values for multiple comparison problems", Technical Report.

[292] Genz, A. (2000), "Numerical computation of bivariate and trivariate normal probabilities", Technical Report.

[293] Genz, A. (2004), "Numerical computation of rectangular bivariate and trivariate normal and t-probabilities", *Statistics and Computing*, **14**, 251-260.

[294] Genz, A. & Bretz, F. (1999), "Methods for the computation of multivariate t-probabilities", Technical Report.

[295] Genz, A. & Bretz, F. (1999), "Numerical computation of multivariate t-probabilities with application to power calculation of multiple contrasts", Technical Report.

[296] Genz, A. & Bretz, F. (2002), "Methods for the computation of multivariate t-probabilities", *Journal of Computational and Graphical Statistics*, **11**, 950-971.

[297] Genz, A. & Bretz, F. (2009), *Computation of multivariate normal and t-probabilities*, Lecture Notes in Statistics, Heidelberg: Springer Verlag.

[298] Genz, A. & Kass, R. (1996), "Subregion adaptive integration of functions having a dominant peak", *J. of Computational and Graphical Stat.*,

[299] Genz, A. & Kwong, K.S. (1999), "Numerical evaluation of singular multivariate normal distributions", Technical Report.

[300] Genz, A. & Monahan, J. (1996), "Stochastic integration rules for infinite regions", *SIAM Journal Scientific Computation*,

[301] George, J.A. & Liu, J.W. (1981), *Computer Solution of Large Sparse Positive Definite Systems*, New Jersey: Prentice-Hall.

[302] George, J.A., Gilbert, J.R., & Liu, J.W.H. (1993), *Graph Theory and Sparse Computations*, Springer Verlag, New York.

[303] Ghali, W.A., Quan, H., Brant, R., van Melle, G., Norris, C.M., Faris, P.D., Galbraith, P.D. & Knudtson, M.L. (2001), "Comparison of 2 methods for calculating adjusted survival curves from proportional hazards model", *Journal of American Medical Association*, **286**, 1494-1497.

[304] Ghosh, S. & Teschmacher, L. (2002), "Comparisons of search designs using search probabilities", *Journal of Statistical Planning and Inference*, **104**, 439-458.

[305] Ghysels, E. and Hall, A. (1990), "A Test for Structural Stability of Euler Conditions Parameters Estimated via the Generalized Method of Moments Estimator," *International Economic Review*, **31**, 355-364.

[306] Gifi, A., (1990) *Nonlinear Multivariate Analysis*, Chichester: Wiley, 1990.

[307] Gilbert, J.R, Ng, E., & Peyton, B.W. (1994), "An efficient algorithm to compute row and column counts for sparse Cholesky factorization", *SIAM J. Matrix Anal. Appl.*, **15**, 1075-1091.

[308] Gill, P.E. , Murray, W., Ponceleon, D.B., & Saunders, M.A. (1992), "Preconditioners for indefinite systems arising in optimization", *SIAM J. on Matrix Analysis and Applications*, **13**, 292-311.

[309] Gill, E.P., Murray, W., Saunders, M.A., & Wright, M.H. (1983), "Computing Forward-Difference Intervals for Numerical Optimization", *SIAM Journal on Scientific and Statistical Computation*, **4**, 310-321.

[310] Gill, E.P., Murray, W., Saunders, M.A., & Wright, M.H. (1984), "Procedures for Optimization Problems with a Mixture of Bounds and General Linear Constraints", *ACM Trans. Math. Software*, **10**, 282-298.

[311] Gill, P.E., Murray, W., & Wright, M.H. (1981), *Practical Optimization*, Academic Press, New York.

[312] Gini, C. (1912), "Variabilità é mutabilita, contributo allo studio delle distribuzioni e delle relazioni statistiche", *Studi Economico - Giuridici della R. Universitá di Cagliari*, **3**, 3-159.

[313] Gleason, J.R. (1988), "Algorithms for balanced bootstrap simulations", *American Statistician*, **42**, 263-266.

[314] Goano, M. (1995), "Algorithm 745: Computation of the Complete and Incomplete Fermi-Dirac Integral", *ACM TOMS*, **21**, 221-232.

[315] Goffe, W.L., Ferrier, G.D., & Rogers, J. (1994): "Global optimization of statistical functions with simulated annealing"; *Journal of Econometrics*, **60**, 65-99.

[316] Gold, C., Holub, A. & Sollich, P., (2005), "Bayesian approach to feature selection and parameter tuning for support vector machine classifiers", *Neural Networks*, **18(5-6)**, 693-701.

[317] Goldberg, D. E. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading: Addison-Wesley.

[318] Goldberg, D. E., & Richardson, J. (1987), "Genetic Algorithms with Sharing for Multimodal Function Optimization", in: *Genetic Algorithms and their Applications*, Proceedings of the Second International Conference on Genetic Algorithms, 41-49.

[319] Goldfarb, D. & Idnani, A. (1983): "A numerically stable dual method for solving strictly convex quadratic programs"; *Mathematical Programming*, **27**, 1-33.

[320] Goldfeld, S.M., Quandt, R.E., & Trotter, H.F. (1966), "Maximisation by quadratic hill-climbing", *Econometrica*, **34**, 541-551.

[321] Golub, G., & Reinsch, C. (1970), "Singular value decomposition and least squares solution", *Numerische Mathematik*, **14**, 403-420.

[322] Golub, G., & Van Loan, C.F. (1980), "An analysis of the total least squares problem", *SIAM Journal Numerical Analysis*, 883-893.

[323] Golub, G., & Van Loan, C.F. (1989), *Matrix Computations*, John Hopkins University Press, 2nd ed., Baltimore, MD.

[324] Gonin, R. & Money, A.H. (1989), *Nonlinear L_p-norm Estimation*, New York: M. Dekker, Inc.

[325] Goodall, C. (1983), "M-estimators of location: An outline of theory"; in: Hoaglin, D.C., Mosteller, F., & Tukey, J.W.: "Understanding Robust and Exploratory Data Analysis", New York: John Wiley & Sons.

[326] Goodman, L.A. (1965), "On simultaneous confidence intervals for multinomial proportions", *Technometrics*, **7**, 247-254.

[327] Goodman, L.A. (1985), "The analysis of cross-classified data having ordered and/or unordered categories: Association models, correlation models, and asymmetry models for contingency tables with or without missing entries", *The Annals of Statistics*, **13**, 10-69.

[328] Goodnight, J.H. (1979), "A Tutorial on the SWEEP Operator", *The American Statistician*, **33**, 149-158.

[329] Gould, N.I.M. (1986), "On the accurate determination of search directions for simple differentiable penalty functions", *IMA Journal of Numerical Analysis*, **6**, 357-372.

[330] Gould, N.I.M. (1989), "On the convergence of a sequential penalty function method for constrained minimization", *SIAM Journal Numerical Analysis*, **26**, 107-128.

[331] Gower, J.C. (1971), "A general coefficient of similarity and some of its properties", Biometrics, **27**, 857-871.

[332] Grassman, R., Gramacy, R.B., & Sterratt, D.C. (2011), *Package 'geometry'*; `http://geometry.r-forge.r-project.org/`.

[333] Graybill, F.A. (1969), *Introduction to Matrices with Applications in Statistics*, Belmont, CA: Wadsworth, Inc.

[334] Green, D. & Swets, J. (1996), *Signal Detection Theory and Psychophysics*, New York: John Wiley, 45-49.

[335] Greenbaum, A. (1997), *Iterative Methods for Solving Linear Systems*, Philadelphia: SIAM.

[336] Grubbs, F. E. (1969), "Procedures for detecting outlying observations in samples", *Technometrics*, **11**, 1-21.

[337] Gupta, A. & Avron, H. (2000, 2013), *WSMP: Watson Sparse Matrix Package*, *Part I - direct solution of symmetric systems*, *Part II - direct solution of general systems*, *Part III - iterative solution of sparse systems*, version 13.11, IBM Research Division, 1101 Kitchawan Road, Yorktown Heights, NY 10598 `http://www.research.ibm.com/projects/wsmp`

[338] Guttman, L. (1953), "Image theory for the structure of quantitative variates", *Psychometrika*, **18**, 277-296.

[339] Guttman, L. (1957), "Empirical Verification of the Radex Structure of Mental Abilities and Personality Traits", *Educational and Psychological Measurement*, **17**, 391-407.

[340] Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002), "Gene selection for cancer classification using support vector machines", *Machine Learning*, **46**, 389-422, 2002.

[341] Guyon, I., & Elisseeff, A. (2003), "An introduction to variable and feature selection", *Journal of Machine Learning Research*, **3**, 1157-1182, 2003.

[342] Hägglund, G. (1982), "Factor Analysis by Instrumental Variable Methods", *Psychometrika*, **47**, 209-222.

[343] Hahsler, M. & Hornik, K. (2013), "TSP - Infrastructure for the traveling salesperson problem", Technical Report, CRAN.

[344] Hald, A. (1952), *Statistical Theory with Engineering Applications*, New York: J. Wiley.

[345] Hald, J. & Madsen, K. (1981), "Combined LP and Quasi-Newton Methods for Minimax Optimization", *Mathematical Programming*, **20**, 49-62.

[346] Hald, J. & Madsen, K. (1985), "Combined LP and Quasi-Newton Methods for Nonlinear l_1 Optimization", *SIAM Journal Numerical Analysis*, **20**, 68-80.

[347] Hall, P. (1992), *The Bootstrap and Edgeworth Expansion*, New York: Springer-Verlag.

[348] Hamers, B., Suykens, J.A.K. & Vandevalle, J. (2002), "Compactly supported RBF kernels for sparsifying the Gram matrix in LS-SVM regression models", *Proceedings ICANN 2002*, Madrid, Spain; 720-726. Technical Report, Kath. University of Leeuven.

[349] Hamilton, J.D. (1994) *Time Series Analysis*, Princeton: Princeton University Press.

[350] Hanley, J.A. & McNeil, B.J. (1982), "The meaning and use of the area under a receiving operating characteristic (ROC) curve", *Radiology*, **143**, 29-36.

[351] Hansen, L.P. (1982), "Large Sample Properties of Generalized Method of Moments Estimators," *Econometrica*, **50**,1029-1054.

[352] Hansen, L.P. & Singleton, K.J. (1982), "Generalized Instrumental Variables Estimation of Nonlinear Rational Expectations Models," *Econometrica*, **50**,1269-1280.

[353] Hansen, P. R. (2005), "A test for superior predictive ability"; *Journal of Business and Economic Statistics*, **23**, 365-380. *Mathematical Programming*, **27**, 1-33.

[354] Harbison, S.P., & Steele, G.L. (1984), *A C Reference Manual*, Prentice Hall, Englewood Cliffs, NJ.

[355] Hardouin, J.-B. (2007), "Non parametric Item Response Theory with SAS and Stata"; *JSS*, 2007.

[356] Hardouin, J.-B. & Mesbah, M. (2007), "The SAS macro-program Item Response Theory Models"; *Communications in Statistics - Simulation and Computation*, **36**, 437-453.

[357] Harman, H.H. (1976), *Modern Factor Analysis*, Chicago: University of Chicago Press.

[358] Hartigan, J. A. (1975), *Clustering Algorithms*, New York: John Wiley & Sons.

[359] Hartmann, W. (1979), *Geometrische Modelle zur Analyse empirischer Daten*, Berlin: Akademie Verlag.

[360] Hartmann, W. (1991), *The NLP Procedure: Extended User's Guide*, Releases 6.08 and 6.10; SAS Institute Inc., Cary, N.C.

[361] Hartmann, W. (1992), *Nonlinear Optimization in IML*, Releases 6.08, 6.09, 6.10; Technical Report, Cary, N.C.: SAS Institute Inc.

[362] Hartmann, W. (1994), "Using PROC NLP for Risk Minimization in Stock Portfolios", Technical Report, Cary, N.C.: SAS Institute Inc.

[363] Hartmann, W. (1995), L_1 *Norm Minimization and Robust Regression - New in IML Release 6.11*, Technical Report, Cary, N.C.: SAS Institute Inc.

[364] Hartmann, W. (1995), *The CALIS Procedure: Extended User's Guide*, Release 6.11; SAS Institute Inc., Cary, N.C.

[365] Hartmann, W. (2003), "User Guide for PROC DMNEURL"; Technical Report, Cary: SAS Institute.

[366] Hartmann, W. (2005), "Resampling methods in structural equations", in: A. Maydeu-Olivares & J.J. McArdle: *Comtemporary Psychometrics (Festschrift for Roderick P. McDonald)*, Mahwah NJ: Laurence Erlbaum.

[367] Hartmann, W. (2007), "Tensor and List Operations in CMAT"; Technical Report, CMAT, Heidelberg.

[368] Hartmann, W. (2007), "Olvi's Matlab Algorithms in CMAT"; Technical Report, CMAT, Heidelberg.

[369] Hartmann, W. (2011), "Automatic Model Improvement usinf the `cfa` Function in CMAT"; Technical Report, CMAT, Heidelberg.

[370] Hartmann, W. (2015), "Difference in p values by PROC LOGISTIC and `elrm` in R"; Technical Report, CMAT, Heidelberg.

[371] Hartmann, W. (2015), "Exact logistic Regression for small Samples"; Technical Report, CMAT, Heidelberg.

[372] Hartmann, W. (2015), "Data Objects in CMAT"; Technical Report, CMAT, Heidelberg.

[373] Hartmann, W.M., & R.E. Hartwig (1996), "Computing the Moore-Penrose Inverse for the Covariance Matrix in Constrained Nonlinear Estimation", *SIAM Journal on Optimization*, **6**, p. 727-747.

[374] Hartmann, W. M., & So, Y. (1995), "Nonlinear Least-Squares and Maximum-Likelihood Estimation Using PROC NLP and SAS/IML", Computer Technology Workshop, American Statistical Association, Joint Statistical Meeting, Orlando, 1995.

[375] *Harwell Subroutine Library: Specifications*, Vol. 1 and 2 (Release 11), ed. 2, Harwell Laboratory, Oxfordshire, UK.

[376] Hastie, T., & Tibshirani, R., (2004), "Efficient quadratic regularization for expression arrays", *Biostatistics*, **5**, 329-340.

[377] Hawkins, D.M. (1994), "The feasible solution algorithm for least trimmed squares regression", *Computational Statistics and Data Analysis*, **17**, p. 185-196.

[378] Hawkins, D.M. (1994), "The feasible solution algorithm for the minimum covariance determinant estimator in multivariate data", *Computational Statistics and Data Analysis*, **17**, p. 197-210.

[379] Hawkins, D.M. & Kass, G.V. (1982), "Automatic Interaction Detection", in Hawkins, D.M. (ed.): *Topics in Applied Multivariate Analysis*, 267-302, Cambridge Univ Press: Cambridge.

[380] Hawkins, D.M., Bradu, D., & Kass, G.V. (1984), "Location of several outliers in multiple regression data using elemental sets", *Technometrics*, **26**, 197-208.

[381] Haymond, R.E., Jarvis, J.P. & Shier, D.R. (2013), "Algorithm 613: Minimum spanning tree for moderate integer weights", *ACM TOMS*, **10**, 108-110.

[382] Hedeker, D. (2012), "MIXREGLS: a Fortran program for mixed-effects location scale analysis", *JSS*.

[383] Hedeker, D. & Gibbons, R.D. (1996), "MIXOR: a computer program for mixed-effects ordinal regression analysis"; *Computer Methods and Programs in Biomedicine*, **49**, 157-176.

[384] Hedeker, D. (1999), "MIXNO: a computer program for mixed-effects nominal regression"; *Journal of Statistical Software*, **4**.

[385] Hedeker, D., Demirtas, H., & Mermelstein, R.J. (2009), "A mixed ordinal location scale model for analysis of Ecological Momentary Assessment (EMA) data"; *Statistics and its Interface*, **2**, 391-401.

[386] Hedeker, D., Mermelstein, R.J., & Demirtas, H. (2008), "An application of a mixed-effects location scale model for analysis of Ecological Momentary Assessment (EMA) data"; *Biometrics*, **64**, 627-634.

[387] Hedeker, D., Mermelstein, R.J., & Demirtas, H. (?), "Modeling between- and within-subject variance in Ecological Momentary Assessment (EMA) data using mixed-effects location scale models"; submitted.

[388] Hegger, R., Kantz, H. & Schreiber, T. (1999), "Practical implementation of nonlinear time series methods: The TISEAN package", *CHAOS*, **9**, 413

[389] Hemker, B.T., Sijtsma, K., & Molenaar, I.W. (1995), "Selection of unidimensional scales from a multidimensional item bank in the polytomous Mokken IRT model"; *Applied Psychological Measurement*, **19**, 337-352.

[390] Heiser, W. J. (1981), *Unfolding Analysis of Proximity Data*; Leiden: Karsten Druckers, PhD Thesis.

[391] Hellakalek, P. & Wegenkittel, S. (2003), "Empirical evidence concerning AES"; *ACM Transactions on Modeling and Computer Simulation*, **13**, 322-333.

[392] Helsgaun, K. (2000), "An effective implementation of the Lin-Kernighan Traveling Salesman Heuristic", *European Journal of Operational Research*, **126**, 106-130.

[393] Helsgaun, K. (2006), "An effective implementation of K-opt moves for the Lin-Kernighan TSP Heuristic", *Datalogiske Skrifter*, **109**, Roskilde University, Denmark.

[394] Henze, N. & Zirkler, B. (1990), "A class of invariant consistent tests for multivariate normality", *Communications in Statistics: Theory and Methods*, **19**,3595-3617.

[395] Herbrich, R. (2002), *Learning Kernel Classifiers: Theory and Algorithms*, Cambridge and London: MIT Press.

[396] Higham, N.J. (1988), "FORTRAN codes for estimating the one-norm of a real or complex matrix with applications to condition estimation", *ACM Trans. Math. Soft.*, **14**, pp. 381-396.

[397] Hirjim, K.F. (2006), *Exact Analysis of Discrete Data*, New York: Chapman and Hall, CRC.

[398] Hjorth, J.S.U. (1994), *Computer Intensive Statistical Methods*, London: Chapman & Hall.

[399] Hoaglin, D.C., Mosteller, F., & Tukey, J.W. (1983), "Understanding Robust and Exploratory Data Analysis", New York: John Wiley & Sons.

[400] Hochberg, Y. (1988), "A sharper Bonferroni procedure for multiple tests of significance", *Biometrika*, **75**, 800-803.

[401] Hochberg, Y. & Tamhane, A. C. (1987), *Multiple Comparison Procedures* New York: Wiley.

[402] Hochreiter, S., Clevert, D.-A., & Obermayer, K. (2006), "A new summarization method for affymetrix probe level data"; *Bioinformatics*, **22**, 943-949.

[403] Hock, W. & Schittkowski, K. (1981), *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems 187, Springer Verlag, Berlin-Heidelberg-New York.

[404] Hodrick, R. J. & Prescott, E. C. (1997), "Postwar US business cycles: An empirical investigation"; *Journal of Money, Credit, and Banking*, **29**, 1-16.

[405] Hoegaerts, L., Suykens, J.A.K., Vandewalle, J., & De Moor, B., (2003), "Kernel PLS variants for regression", in *Proc. of the 11th European Symposium on Artificial Neural Networks*, Bruges, Belgium; 203-208. Technical Report, Kath. University of Leeuven.

[406] Holm, S. (1979), "A simple sequentielle rejective multiple test procedure", *Scandinavian Journal of Statistics*, **6**, 65-70.

[407] Hommel, G. (1988), "A stagewise rejective multiple test procedure based on a modified Bonferroni test", *Biometrika*, **75**, 383-386.

[408] Horan, C. B. (1969), "Multidimensional scaling: Combining observations when individuals have different perceptual structures"; *Psychometrika*, **34**, 139-165.

[409] Horn, J. L. & Engstrom, R. (1979), "Cattell's scree test in relation to Bartlett's Chi-square test and other observations on the number of factors problem; *Multivariate Behavioral Research*, **14**, 283-300.

[410] Horn, R.A. & Johnson, C.R. (1985, 1996), *Matrix Analysis*, Cambridge University Press, Cambridge.

[411] Hoyer, P. O. (2004), "Non-negative matrix factorization with sparseness constraints"; *Journal of Machine Learning Research*, **5**, 1457-1469.

[412] Hsu, C.-W. & Lin, C.-J. (1999), "A Simple Decomposition Method for Support Vector Machines", Technical Report, Dept. of Computer Science and Information Engineering; National University of Taiwan.

[413] Huang, C. (2011), "SAS vs. R in data mining", `http://www.sasanalysis.com/2011/01/why-sas-is-more-useful-t`

[414] Huber, P. (1981), *Robust Statistics*, New York: John Wiley & Sons.

[415] Huber, P. J. (1985), "Projection pursuit", *The Annals of Statistics*, **13**, 435-475.

[416] Huber, W., Heydebreck, A.v., Sueltmann, H., Poustka, A., & Vingron, M. (2002), "Variance stabilization applied to microarray data calibration and to quantification of differential expression"; *Bioinformatics*, **18**, S96-S106.

[417] Huber, W., Heydebreck, A.v., Sueltmann, H., Poustka, A., & Vingron, M. (2003), "Parameter estimation for the calibration and variance stabilization of microarray data"; *Statistical Applications in Genetics and Molecular Biology*, **2**, No. 1, Article 3.

[418] Huddleston, H.F., Claypool, P.L., & Hocking, R.R. (1970), "Optimal sample allocation to strata using convex programming"; *Applied Statistics*, **19**, 273-278.

[419] Hyvaerinen, A., Karhunen, J. & Oja, E. (2001) *Independent Component Analysis*, New York: J. Wiley & Sons.

[420] Hwang, Y.T. & Wang, C.C. (2009), "Assessing multivariate normality based on Shapiro-Wilk test", *Journal of the Chinese Statistical Association*, **47**, 143-158.

[421] Iglewicz, B. (1983), "Robust scale estimators and confidence intervals for location"; in: Hoaglin, D.C., Mosteller, F., & Tukey, J.W.: "Understanding Robust and Exploratory Data Analysis", New York: John Wiley & Sons.

[422] James, L.R., Mulaik, S.A., & Brett, J.M. (1982), *Causal Analysis: Assumptions, Models, and Data*, Beverly Hills: Sage Publications, Inc.

[423] Janert, P. K. (2009), *Gnuplot in Action: Understanding data with graphs*, Greenwich CT: Manning Publications Co.

[424] Jarque, C. M. & Bera, A. K. (1987), "A test for normality of observations and regression residuals", *International Statistical Review*, **55**, 163-172.

[425] Jenkins, M.A. & Traub, J.F. (1972), "Zeros of a complex polynomial", ACM.

[426] Jenkins, M.A. & Traub, J.F. (1975), "Principles for testing polynomial zerofinding programs", ACM TOMS, **1**, 26-34.

[427] Jennrich, R. I. (1973), "Standard errors for obliquely rotated factor loadings"; *Psychometrika*, **38**, 593-604.

[428] Jennrich, R. I. (1974), "Simplified formulae for standard errors in maximum likelihood factor analysis"; *British Journal of Math. and Statist. Psychology*, **27**, 122-131.

[429] Jennrich, R.I. (1987), "Tableau Algorithms for Factor Analysis by Instrumental Variable Methods", *Psychometrika*, **52**, 469-476.

[430] Jennrich, R.I. & Schluchter, M.D. (1986), "Unbalanced repeated-measures models with structured covariance matrices", *Biometrics*, **42**, 805-820.

[431] Jensen, D.R. & Ramirez, D.E. (1998), "Detecting outliers with Cook's D_I statistics", *Computing Science and Statistics*, **29(1)**, 581-586.

[432] Joachims, T. (1999), "Making large-scale SVM learning practical", in B. Schölkopf, C.J.C. Burges, and A.J. Smola (eds), *Advances in Kernel Methods: Support Vector Learning*, Cambridge: MIT Press.

[433] Joe, H. & Xu, J. (2007), "The estimation method of inference for margins for multivariate models"; Technical Report.

[434] Jöreskog, K.G. (1963), *Statistical Estimation in Factor Analysis*, Stockholm: Almqvist & Wicksell.

[435] Jöreskog, K.G. (1969), "Efficient estimation in image factor analysis", Psychometrika, **34**, 51-75.

[436] Jöreskog, K.G. (1973), "A general method for estimating a linear structural equation system", in *Structural Equation Models in the Social Sciences*, eds. A.S. Goldberger & O.D. Duncan, New York: Academic Press.

[437] Jöreskog, K.G. (1978), "Structural Analysis of Covariance and Correlation Matrices", *Psychometrika*, **43**, 443-477.

[438] Jöreskog, K.G. (1982), "Analysis of Covariance Structures", in *A Second Generation of Multivariate Analysis*, ed. C. Fornell, New York: Praeger Publishers.

[439] Jöreskog, K.G. & Sörbom, D. (1979), *Advances in Factor Analysis and Structural Equation Modeling*, Cambridge MA: Abt Books.

[440] Jöreskog, K.G. & Sörbom, D. (1988), *LISREL 7: A Guide to the Program and Applications*, SPSS Inc., Chicago, Illinois.

[441] Johnson, K., Mandal, A. & Ding, T. (2007), "Software for implementing the sequential elimination algorithm of level combination algorithm"; JSS 2007.

[442] Johnson, M. E. (1987), *Multivariate Statistical Simulation*, New York: John Wiley & Sons.

[443] Jonker, R. & Volgenant, A. (1983), "Transforming asymmetric into symmetric traveling salesman problems", *Operations Reseatrch Letters*, **2**, 161-163.

[444] Jonker, R. & Volgenant, A. (1987), "A shortest augmenting path algorithm for dense and sparse linear assignment problems", *Computing*, **38**, 325-340.

[445] Kahaner, D., Moler, C. & Nash, S. (1989), *Numerical Methods and Software*, Prentice Hall, Englewood Cliffs, NJ.

[446] Kantz, H. (1994), "A robust method to estimate the maximal Lyapunov exponent of a time series", *Phys. Lett.*, A 185, 77

[447] Kantz, H. & Schreiber, T. (2004), *Nonlinear Time Series Analysis*, 2nd edition, Cambridge: University Press.

[448] Kass, G.V. (1980), "An exploratory technique for investigating large quantities of categorical data", *Applied Statistics*, **29**, 119-127.

[449] Kaufman, L. & Rousseeuw, P.J. (1990), *Finding Groups in Data*, New York: John Wiley & Sons.

[450] Kay, S. M. & Marple, S. L. Jr. (1981), "Spectrum analysis - a modern perspective"; *Proceedings of the IEEE*, **69**, 1380-1419.

[451] Keerthi, S.S., Shevade, S.K., Bhattacharyya, C, & Murthy, K.R.K. (1999a), "A fast iterative nearest point algorithm for support vector machine classifier design", Technical Report TR-ISL-99-03, Department of CSA, Bangalore, India.

[452] Keerthi, S.S., Shevade, S.K., Bhattacharyya, C, & Murthy, K.R.K. (1999b), "Improvements of Platt's SMO algorithm for SVM classifier design", Technical Report CD-99-14, Dep. of Mechanical Production and Engineering, National University of Singapore.

[453] Kellerer, H., Pferschy, U., & Pisinger, D. (2004), *Knapsack Problems*, Berlin, Heidelberg: Springer Verlag.

[454] Kennedy, W.J. & Gentle, J,E. (1988), *Statistical Computing*, New York: Marcel Dekker.

[455] Kernighan, B.W., & Ritchie, D.M. (1978), *The C Programming Language*, Prentice Hall, Englewood Cliffs, NJ.

[456] Keuls, M. (1952), "The use of the studentized range in connection with an analysis of variance", *Euphytica*, **37**, 112-122.

[457] Kim, M. (1993), "Forecasting the Volatility of Financial Markets: ARCH/GARCH Models and the AUTOREG Procedure", in *Proceedings of the Eighteenth Annual SAS Users Group International Conference*, pp. 304-312, Cary: SAS Institute Inc.

[458] Kleinbaum, D.G. (1998), Kupper, L.L., Muller, K.E., & Nizam, A. (1998), *Applied Regression Analysis and Other Multivariate Methods*, North Scituate, MA: Duxbury Press.

[459] Klugkist, I., Laudy, O., & Hoijtink, H. (2005), "Inequality constrained analysis of variance, A Bayesian Approach", *Psychological Methods*, **10**, 477-493.

[460] Klugkist, I. & Hoijtink, H. (2007), "The Bayes factor for inequality and about equality constrained models", *Computational Statistics and Data Analysis*, **51**, 6367-6379.

[461] Knuth, D.E. (1986), *The TEXbook*, Seventh Printing, Addison-Wesley, Reading, MA.

[462] Knuth, D.E. (1973), *The Art of Computer Programming; Vol.1: Fundamental Algorithms*, *Vol.2: Seminumerical Algorithms*, *Vol.3: Sorting and Searching*, Addison-Wesley, Reading, MA.

[463] Kohonen, T. (2001), *Self-Organizing Maps*, Springer Verlag, Berlin.

[464] Kolda, T.G. & O'Leary, D.P. (1998), "A semidiscrete matrix decomposition for latent semantic indexing in information retrieval"; *ACM Trans. Inf. Syst.*, **16**, 322-346.

[465] Kolda, T.G. & O'Leary, D.P. (1999a), "Latent semantic indexing via a semi-discrete matrix decomposition", in *The Mathematics of Information Coding. Extraction and Distribution*, Vol 107 of the *IMA Volumes in Mathematics and Its Applications*, pp. 73-80, Springer Verlag.

[466] Kolda, T.G. & O'Leary, D.P. (1999b), *Computation and Uses of the Semidiscrete Matrix Decomposition*, Technical Report, Sandia National Laboratories, Livermore, CA.

[467] Kolmogorov, A. (1933), "Sulla determinazione empirica di una legge di distributione", *Giornale dell' Instituto Italiano degli Attuari*, **4**, 83-91.

[468] Krane, W.R. & McDonald, R.P. (1978), "Scale invariance and the factor analysis of correlation matrices", *British Journal of Mathematical and Statistical Psychology*, **31**, 218-228.

[469] Kreft, I. & de Leeuw, J. (1998), *Introducing Multilevel Modeling*, Beverly Hills, CA: SAGE Publications, Inc.

[470] Kruskal, J.B. (1964a), "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis", *Psychometrika*, **29**, 1-27.

[471] Kruskal, J.B. (1964b), "Nonmetric multidimensional scaling: A numerical method", *Psychometrika*, **29**, 115-129.

[472] Kruskal, J. B., Young, F. W. & Seery, J. B. (1978), "How to use KYST, a very flexible program to do multidimensional scaling and unfolding"; Technical Report, Murray Hill: Bell Laboratories.

[473] Kruskal, W.H. & Wallis, W.A. (1952), "Use of ranks in one-criterion variance analysis", JASA, **47**, 583-621.

[474] Kruskal, W.H. (1957), "Historical Note on the Wilcoxon unpaired two-sample test", JASA, **52**, 356-360.

[475] Kugiumtzis, D. (2002), "Surrogate Data Test for Nonlinearity using Statically Transformed Autoregressive Process", *Physical Review E*, **66**, 025201.

[476] Kugiumtzis, D. (2009), "Measures of analysis of time series toolkit (MATS)", paper submitted to JSS.

[477] Kuiper, R.M., Klugkist, I. & Hojtink, H. (2007), "A Fortran 90 Program for Confirmatory Analysis of Variance", *JSS* 422 .

[478] Kursa, M.B. & Rudnicki, W.R. (2010), "Feature Selecion with the Boruta Package"; *JSS*, 2010.

[479] Lambert, Z.V., Wildt, A.R. & Durand, R.M. (1990), "Assessing sampling variation relative to number-of-factors criteria"; *Educational and Psychological Measurement*, **50**, 253-257.

[480] Lambert, Z.V., Wildt, A.R. & Durand, R.M. (1991), "Approximating confidence intervals for factor loadings"; *Multivariate Behavioral Research*, **26**, 412-434.

[481] Lamport, L. (1986), *LaTeX - A Document Preparation System - User's Guide and Reference Manual*, Fifth Printing, Addison-Wesley, Reading, MA.

[482] "Using SAS GENMOD procedure to fit diagonal class models to square contingency tables having ordered categories"; *Proceedings of the Midwest SAS Users Group*, 149-160.

[483] Lawal, H.B. & Sundheim, R.A. (2002), "Generating factor variables for asymmetry, non-independence, and skew-symmetry models in square contingency tables using SAS", *JSS*, 2002.

[484] Lawless, J.F. (1982), *Statistical Models and Methods for Lifetime Data*, New York: John Wiley & Sons, Inc.

[485] Lawley, D.N. & Maxwell, A.E. (1971), *Factor Analysis as a Statistical Method*, New York: American Elsevier Publishing Company.

[486] Lawson, C.L. & Hanson, R.J. (1974), *Solving Least Squares Problems*, Englewood Cliffs, NJ: Prentice Hall.

[487] Lawson, C.L. & Hanson, R.J. (1995), *Solving Least Squares Problems*, Philadelphia, PA: SIAM.

[488] Lawson, C.L.,Hanson, R.J., Kincaid, D.R., & F.T. Krogh (1979), "Basic linear algebra subprograms for Fortran usage", *ACM Transactions on Mathematical Software*, **5**, pp. 308-323 and 324-325.

[489] L'Ecuyer, P. (1996a), "Combined Multiple Recursive Random Number Generators", *Operations Research*, **44**, 816-822.

[490] L'Ecuyer, P. (1996b), "Maximally equidistributed combined Tausworthe generators", *Math. of Comput.*, **65**, 203-213.

[491] L'Ecuyer, P. (1999), "Tables of maximally equidistributed combined LFSR generators", *Math. of Comput.*, **68**, 261-269.

[492] Ledoit, O. & Wolf, M. (2003a), "Improved estimation of the covariance matrix of stock returns with an application to portfolio selection"; *Journal of Empirical Finance*, **10**, 603-621.

[493] Ledoit, O. & Wolf, M. (2003b), "Honey I shrank the sample covariance matrix"; Technical Report.

[494] Lee, W. and Gentle, J.E. (1986), "The LAV Procedure", *SUGI Supplemental Library User's Guide*, Cary: SAS Institute, Chapter 21, pp. 257-260.

[495] Lee, D.D. & Seung, S. (1999), "Learning the parts of objects by non-negative matrix factorization"; *Nature*, **401**, 788-791.

[496] Lee, D.D. & Seung, S. (2001), "Algorithms for non-negative matrix factorization"; *Advances in Neural Information Processing Systems*, **13**, 556-562.

[497] Lee, J.Y., Soong, S.J., Shaw, H.M., Balch, C.M., McCarthy, W.H. & Urist, M.M. (1992), "Predicting survival and recurrence in localized melanoma: A multivariate approach"; *World Journal of Surgery*, **16**, 191-195.

[498] Lee, J., Yoshizawa, C., Wilkens, L., & Lee, H.P. (1992), "Covariance adjustment of survival curves based on Cox's proportional hazards regression model", *Comput. Appl. Biosc.*, **8**, 23-27.

[499] Lee, S.Y. (1985),"On Testing Functional Constraints in Structural Equation Models", *Biometrika*, **72**, 125-131.

[500] Lee, S.Y. & Jennrich, R.I. (1979), "A Study of Algorithms for Covariance Structure Analysis with Specific Comparisons Using Factor Analysis", *Psychometrika*, **44**, 99-113.

[501] Lee, Y., Lin, Y., & Wahba, G. (2003), *Multicategory Support Vector Machines, Theory and Application to the Classification of Microarray Data and Satellite Radiance Data*, Technical Report 1064, Dep. of Statistics, Univ. of Wisconsin, 2003.

[502] Lee, Y., Kim, Y., Lee, S. & Koo, J.Y. (2005), *Structured Multicategory Support Vector Machines with Analysis of Variance Decomposition*, Technical Report .

[503] Lee, Y.-Y. & Mangasarian, O. (1999), *SSVM: A smooth support vector machine*, Technical Report 99-03, Computer Sciences Dept., University of Wisconsin, Madison.

[504] Lee, Y.-Y. & Mangasarian, O. (2000), *RSVM: Reduced support vector machines*, Technical Report 00-07, Computer Sciences Dept., University of Wisconsin, Madison.

[505] Lehoucq, R.B., Sorensen, D.C., Yang, C. (1998), *ARPACK Users' Guide*, Philadelphia: SIAM.

[506] Leong, P.H.W., Zhang, G., Lee, D.U., Luk, W., & Villasenor, J.D. (2005), "A comment on the implementation of the ziggurat method"; JSS **12**.

[507] Lewis,J.G. (1977), "Algorithms for Sparse Matrix Eigenvalue Problems", Report STAN-CS-77-595, Computer Science Department, Stanford University, Stanford, California, March 1977.

[508] Li, K.C. (1991), "Sliced inverse regression for dimension reduction", *JASA*, **86**, 316-342.

[509] Li, K.C. (1992), "On principal Hessian directions for data visualization and dimension reduction: Another application of Stein's lemma", *JASA*, **87**, 1025-1034.

[510] Liaw, A. (2009), *randomForest* package in R.

[511] Liaw, A. & Wiener, M. (2002), "Classification and Regression by randomForest", *R News*, **2**, 3.

[512] Liang, K.Y. & Zeger, S.L. (1986) *Longitudinal data analysis using generalized linear models*, *Biometrika*, **73**, 13-22.

[513] Liebman, J.; Lasdon, L.; Schrage, L.; and Waren, A. (1986), *Modeling and Optimization with GINO*, California: The Scientific Press.

[514] Lin, C.-J. & Moré, J.J. (1999), "Newton's method for large bound constrained problems", *SIAM Journal of Optimization*, **9**, 1100-1127.

[515] Lin, S. (1965), "Computer solutions of the traveling-salesman problem", *Bell Systems Technology*, **44**, 2245-2269.

[516] Lin, S. & Kernighan, B. (1973), "An effective heuristic algorithm for the traveling-salesman problem", *Operations Research*

[517] Lin, Y., Lee, Y., & Wahba, G. (2000), *Support Vector Machines for Classification in Nonstandard Situations*, Technical Report 1016, Dep. of Statistics, Univ. of Wisconsin, 2000.

[518] Lindström, P. & Wedin, P.A. (1984), "A new linesearch algorithm for nonlinear least-squares problems", *Mathematical Programming*, **29**, 268-296.

[519] Liu, H & Wu, T. (2003), "Estimating the Area under a Receiver Operating Characteristic (ROC) Curve for Repeated Measures Design", *JSS*, 2003.

[520] Liu, J. W-H. (1985), "Modification of the minimum degree algorithm by multiple elimination", *ACM Trans. Math. Software*, **11**, 141-153.

[521] Liu, L., Hawkins, D.M., Ghosh, S., & Young, S.S. (2003), "Robust Singular Value Decomposition Analysis of Microarray Data"; *PNAS*, **100**, 13167-13172 (Nov. 11, 2003).

[522] Liu, W., Jamshidian, M., Zhang, Y., Bretz, F., & Han, X. (2007), "Some new methods for the comparison of two regression models", *Journal of Statistical Planning and Inference*, **137**, p. 57-67.

[523] Loevinger, J. (1948), "The technic of homogeneous tests compared with some aspects of scale analysis and factor analysis"; *Psychological Bulletin*, **45**, 507-529.

[524] Long, J.S. (1983), *Covariance Structure Models, an Introduction to LISREL*, Beverly Hills, CA: SAGE Publications, Inc.

[525] Long, J.S. (1997) *Regression Models for Categorical and Limited Dependent Variables*; Beverly Hills, CA: SAGE Publications, Inc.

[526] Lopuhaä, H.P. & Rousseeuw, P.J. (1991), "Breakdown Points of Affine Equivariant Estimators of Multivariate Location and Covariance Estimators", *The Annals of Statistics*, **19**, 229-248.

[527] Lord, F.M., & Novick, M.R. (1968), *Statistical theories of mental test scores*; Reading, MA: Addison-Wesley.

[528] Lucas, J. W. (2003), "Status processes and the institutionalization of women as leaders"; *American Sociological Review*, **68**, 464-480.

[529] Lüscher, M. (1994), "A portable high-quality random number generator for lattice field theory simulations"; *Computer Physics Communications*, **79**, 100-110.

[530] MacKinnon, D.P. (1992), "Statistical simulation in CALIS", *Proceedings of the 17th Annual SAS Users Group International Conference*, 1199-1203, Cary NC: SAS Institute Inc.

[531] Madsen, K. (1975), "Minimax Solution of Non-Linear Equations Without Calculating Derivatives", *Mathematical Programming Study*, **3**, 110-126.

[532] Madsen, K. & Nielsen, H.B. (1990), "Finite algorithms for robust linear regression", *BIT*, **30**, 682-699.

[533] Madsen, K. & Nielsen, H.B. (1993), "A finite smoothing algorithm for linear l1 estimation", *SIAM Journal on Optimization*, **3**, 223-235.

[534] Madsen, K., Nielsen, H.B., & Pinar, M.C. (1995), "A finite continuation algorithm for bound constrained quadratic programming", Technical Report IMM-REP-1995-22, Technical University of Denmark.

[535] Madsen, K., Nielsen, H.B., & Pinar, M.C. (1996), "A new finite continuation algorithm for linear programming", *SIAM Journal on Optimization*, **6**, 600-616.

[536] Madsen, K., Tingleff, O., Hansen, P.C., Owczarz, W. (1990), "Robust Subroutines for Non-Linear Optimization", Technical Report NI-90-06, Technical University of Denmark.

[537] Maiti, S. I. (2009), "Seasonal adjustment in SAS/BASE software - An alternative to PROC X11/X12"; JSS 442, 2009.

[538] Malinowski, E. R. (1991), *Factor Analysis in Chemistry*, New York: John Wiley & Sons.

[539] Malkiel, B. C. (1985), *A Random Walk Down Wall Street*, New York: Norton, Fourth Edition.

[540] Mandal, A., Wu, C.F.J., & Johnson, K. (2006), "SELC: Sequential elimination of level combinations by means of modified genetic algorithms"; *Technometrics*, **48(2)**, 273-283.

[541] Mandal, A., Johnson, K., Wu, C.F.J., & Bornemeier, D. (2007), "Identifying promising compounds in drug discovery: Genetic algorithms and some new statistical techniques"; *Journal of Chemical Information and Modeling*, **47(3)**, 981-988.

[542] Mangasarian, O.L. (1995), *Nonlinear Programming*, Philadelphia: SIAM.

[543] Mangasarian, O.L. & Musicant, D.R (1999), "Successive overrelaxion for support vector machines", *IEEE Transactions on Neural Networks*, **10**, 1032-1037.

[544] Mangasarian, O.L. & Musicant, D.R (2000a), "Active Support Vector Machine Classification", Technical Report 00-04, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[545] Mangasarian, O.L. & Musicant, D.R (2000b), "Lagrangian Support Vector Machines", Technical Report 00-06, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[546] Mangasarian, O.L. & Thompson, M.E. (2006), "Massive data classification via unconstrained support vector machines", *Journal of Optimization Theory and Applications*. Technical Report 06-07, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[547] Mangasarian, O.L. & Thompson, M.E. (2006), "Chunking for massive nonlinear kernel classification", Technical Report 06-07, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[548] Mangasarian, O.L. & Wild, E.W. (2004), "Feature Selection in k-Median Clustering", Technical Report 04-01, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[549] Mangasarian, O.L. & Wild, E.W. (2006), "Feature Selection for Nonlinear Kernel Support Vector Machines", Technical Report 06-03, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[550] Mann, H. & Whitney, D. (1947), "On a test whether one of two random variables is stochastically larger than the other"; *Annals of Mathematical Statistics*, **18**, 50-60.

[551] Mardia, K. V. (1970), "Measures of multivariate skewness and kurtosis with applications", *Biometrika*, **57**, 519-530.

[552] Mardia, K. V. (1974), "Applications of some measures of multivariate skewness and kurtosis in testing normality and robustness studies", *Sankhya: The Indian Journal of Statistics*, *Series B*, **36**, 115-128.

[553] Mardia, K. V. & Foster, K. (1983), "Omnibus tests of multinormality based on skewness and kurtosis", *Communications in Statistics*, *Part A: Theory and Methods*, **12**, 207-221.

[554] Mardia, K. V., Kent, J.T., & Bibby, J.M. (1979), *Multivariate Analysis*,

[555] Markowitz, H. (1952), "Portfolio Selection", *Journal of Finance*, **7**, 99-91.

[556] Markowitz, H. (1959), *Portfolio Selection: Efficient Diversification of Investments*, New York: John Wiley & Sons, Inc.

[557] Markus, M.T. (1994), *Bootstrap Confidence Regions in Nonlinear Multivariate Analysis*, Leiden: DSWO Press.

[558] Marple, S. L. Jr. (1991), "A fast computational algorithm for the modified covariance method of linear prediction"; *Digital Signal Processing*, **1**, 124-133.

[559] Marsaglia, G., "The Marsaglia Random Number CDROM", with "The Diehard Battery of Tests of Randomness", `www.stat.fsu.edu/pub/diehard`

[560] Marsaglia, G. (1965), "Ratios of normal variables and ratios of sums of uniform variables"; *JASA*, **60**, 193-204.

[561] Marsaglia, G. (2003), "Random number generators", *Journal of Modern Applied Statistical Methods*, **3**, .

[562] Marsaglia, G. (2003), "Xorshift RNGs", *JSS*, 2003.

[563] Marsaglia, G. (2004), "Evaluating the normal distribution function", JSS, 2004.

[564] Marsaglia, G. (2004), "Fast generation of discrete random variables", JSS, 2004.

[565] Marsaglia, G. (2006), "Ratios of normal variables"; *JSS*, 2006.

[566] Marsaglia, G., & Tsang, W.W. (1998), "The Monte Python method for generating Gamma variables", JSS **3**, 1998.

[567] Marsaglia, G., & Tsang, W.W. (2000), "The Ziggurat method for generating random variables", JSS **5**, 2000.

[568] Marsaglia, G. & Tsang, W.W. (2002), "Some difficult to pass tests of randomness", *JSS*, 2002.

[569] Marsaglia, G., Tsang, W.W., & Wang, J. (2003), "Evaluating Kolmogorov's Distribution", JSS, **8**, 2003.

[570] Marsaglia, G., Tsang, W.W., & Wang, J. (2004), "Fast generation of discrete random variables"; JSS, 2004.

[571] Marsaglia, G. & Tsay, L.H. (1985), "Matrices and the structure of random number sequences", *Linear Algebra and its Applications*, **67**, 147-156.

[572] Marsh, H.W., Balla, J.R., & McDonald, R.P. (1988), "Goodness-of-fit indices in confirmatory factor analysis. The effect of sample size", *Psychological Bulletin*, **103**, 391-410.

[573] Martello, S. (1983), "Algorithm 595: An enumerative algorithm for finding Hamiltonian circuits in a directed graph", *ACM TOMS*, **9**, 131-138.

[574] Martello, S. & Toth, P. (1990), *Knapsack Problems: Algorithms and Computer Implementations*, New York: Wiley & Sons.

[575] Mashtare, T. L., Jr. & Hutson, A. D. (2010), "SAS Macro for estimating the standard error of area under the curve with an application to receiver operating curves"; paper submitted to *JSS*.

[576] Masters, G.N. (1982), "A Rasch model for partial credit scoring"; *Psychometrika*, **47**, 149-174.

[577] *MATLAB Reference Guide* (1992), The MathWorks, Inc., Natick MA.

[578] Matsumoto, M. & Nishimura, T. (1998), "Mersenne-Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator"; *ACM Transactions on Modeling and Computer Simultaion*, **8**, 3-30.

[579] May, W.L. & Johnson, W.D. (1997), "A SAS macro for constructing simultaneous confidence intervals for multinomial proportions", *Computer Methods and Programs in Biomedicine*, **53**, 153-162.

[580] May, W.L. & Johnson, W.D. (2000), "Constructing two-sided simultaneous confidence intervals for multinomial proportions for small counts in a large number of cells", *JSS*, 2000.

[581] Maydeu-Olivares, A. (2001), "Multidimensional IRT Modeling of Binary Data: Large Sample Properties of NOHARM Estimates", *J. of Educational and Behavioral Statistics*, **26**, 49-69.

[582] Maydeu-Olivares, A. (2005), "Further empirical results on parametric vs. non-parametric IRT modeling of Likert-type personality data"; *Multivariate Behavioral Research*, **40**, 275-293.

[583] Maydeu-Olivares, A. (2006), "Limited information estimation and testing of discretized multivariate normal structural models"; *Psychometrika*, **71**, 57-77.

[584] Maydeu-Olivares, A., Coffman, D.L. & Hartmann, W. (2007), "Asymptotically distribution-free (ADF) interval estimation of coefficient alpha", *Psychological Methods*, **12**, 157-176.

[585] Maydeu-Olivares, A., & Joe, H. (2005), "Limited and full information information estimation and goodness-of-fit testing in 2^n contingency tables: A unified framework"; *JASA*, **100**, 1009-1020.

[586] Maydeu-Olivares, A., & Joe, H. (2006), "Limited information goodness-of-fit testing in multidimensional contingency tables"; *Psychometrika*, **71**, 713-732.

[587] McArdle, J.J. (1980), "Causal Modeling Applied to Psychonomic Systems Simulation", *Behavior Research Methods & Instrumentation*, **12**, 193-209.

[588] McArdle, J.J. (1988), "Dynamic but Structural Equation Modeling of Repeated Measures Data", in *The Handbook of Multivariate Experimental Psychology*, eds. J.R. Nesselroade and R.B. Cattell, New York: Plenum Press.

[589] McArdle, J.J. & Boker, S.M. (1986), *RAMpath - Path Diagram Software*, Denver: DATA Transforms, Inc.

[590] McArdle, J.J. & McDonald, R.P. (1984), "Some Algebraic Properties of the Reticular Action Model", *Br. J. math. statist. Psychol.*, **37**, 234-251.

[591] MacCallum, R. (1986), "Specification Searches in Covariance Structure Modeling", *Psychological Bulletin*, **100**, 107-120.

[592] McBane, G.C. (2006), "Programs to compute distribution functions and critical values for extreme value ratios for outlier detection"; *JSS*, 2006.

[593] McKean, J.W. and Schrader, R.M. (1987), "Least absolute errors analysis of variance", In: Y. Dodge, ed. *Statistical Data Analysis - Based on L_1 Norm and Related Methods*, Amsterdam: North Holland, 297-305.

[594] McDonald, R.P. (1978), "A Simple Comprehensive Model for the Analysis of Covariance Structures", *Br. J. math. statist. Psychol.*, **31**, 59-72.

[595] McDonald, R.P. (1980), "A Simple Comprehensive Model for the Analysis of Covariance Structures: Some Remarks on Applications", *Br. J. math. statist. Psychol.*, **33**, 161-183.

[596] McDonald, R.P. (1984), "Confirmatory Models for Nonlinear Structural Analysis", in *Data Analysis and Informatics*, III, eds. E. Diday et al., North Holland: Elsevier Publishers.

[597] McDonald, R.P. (1985), *Factor Analysis and Related Methods*, Hillsdale NJ and London: Lawrence Erlbaum Associates.

[598] McDonald, R.P. (1989), "An Index of Goodness-of-Fit Based on Noncentrality", *Journal of Classification*, **6**, 97-103.

[599] McDonald, R.P. (1997), "Normal ogive multidimensional model", in W.J. van der Linden and R. K. Hambleton (Eds.): *Handbook of Modern Item Response Theory*, pp. 257-269, New York: Springer.

[600] McDonald, R.P., & Hartmann, W. (1992), "A Procedure for Obtaining Initial Values of Parameters in the RAM Model", *Multivariate Behavioral Research*, **27**, 57-76.

[601] McDonald, R.P. & Marsh, H.W. (1988), "Choosing a Multivariate Model: Noncentrality and Goodness of Fit", Distributed Paper.

[602] McDonald, R.P. & Mok, M.C. (1995), "Goodness of fit in item response models", *Multivariate Behavioral Research*, **54**, 483-495.

[603] McDonald, R.P., Parker, P.M., & Ishizuka, T. (1993), "A scale-invariant treatment of recursive path models", *Psychometrika*, **58**, 431-443.

[604] McKean, J.W. & Schrader, R.M. (1987), "Least absolute errors analysis of variance", In: Y. Dodge, ed. *Statistical Data Analysis - Based on L_1 Norm and Related Methods*, Amsterdam: North Holland, 297-305.

[605] McLachlan, G.J. (1987), "On bootstrapping the likelihood ratio test statistic for the number of components in a normal mixture", *Applied Statistics*, **36**, 318-324.

[606] McLachlan, G.J. & Krishnan, T. (1997), *The EM Algorithm and Extensions*, New York: John Wiley & Sons.

[607] McMillan, G.P. & Hanson, T. (2005), "SAS Macro BDM for fitting the Dale (1984) regression model to bivariate mordinal response data"; JSS, 2005.

[608] McMillan, G.P., Hanson, T., Bedrick, E. & Lapham, S.C. (2005), "Application of the Bivariate Dale Model to Estimating Risk Factors for Alcohol Consumption Patterns", submitted.

[609] Meeker, W.Q. & Escobar, L.A. (1995), "Teaching about approximate confidence regions based on maximum likelihood estimation", *American Statistician*, **49**, 48-53.

[610] Mehta, C. R. & Patel, N. R. (1986), " Algorithm 643: FEXACT: A Fortran subroutine for Fisher's exact test on unordered $r * c$ contingency tables", *ACM Transactions on mathematical Software*, **12**, 154-161.

[611] Meulman, J. (1982) *Homogoneity Analysis of Incomplete Data*, Leiden: DSWO Press, 1982.

[612] Miha, A., Hayter, J., & Kuriki, S. (2003), "The evaluation of general non-centerd orthant probabilities", *British Journal of the Royal Statistical Society*, Ser. B, **65**, 223-234.

[613] Milan, L. & Whittaker, J. (1995), "Application of the parametric bootstrap to models that incorporate a singular value decomposition"; *Applied Statistics*, **44**, 31-49.

[614] Miles, R.E. (1959), "The complete amalgamation into blocks, by weighted means, of a finite set of real numbers", *Biometrika*, **46**, 317-327.

[615] Miller, A. (2002), *Subset Selection in Regression*, CRC Press, Chapman & Hall, 2nd Edition.

[616] Miller, A.J. (1990), *Subset Selection in Regression*, New York: Chapman and Hall.

[617] Mills T. C. (1990), *Time Series Techniques for Economists*, Cambridge: Cambridge University Press.

[618] *MKS Lex & Yacc*, (1993), Mortice Kern Systems Inc., Waterloo, Ontario CA.

[619] Mokken, R.J. (1971), *A Theory and Procedure of Scale Analysis*; DeGruyter.

[620] Mokken, R.J. (1997), "Nonparametric models for dichotomous responses"; in: W. J. van der Linden and R. K. Hambleton (eds): *Handbook of Modern Item Response Theory* New York: Springer.

[621] Molenaar, I.W. (1997), "Nonparametric models for polytomous responses"; in: W. J. van der Linden and R. K. Hambleton (eds): *Handbook of Modern Item Response Theory* New York: Springer.

[622] Molenberghs, G. & Lesaffre, E. (1994), "Marginal Modeling of Correlated Ordinal Data Using a Multivariate Plackett Distribution", *JASA*, **89**, 633-644.

[623] Momma, M. & Bennett, K.P. (2004), "Constructing orthogonal latent features for arbitrary loss"; Technical Report; Troy: Rensselaer Polytechnic Institute.

[624] Monahan, J.F. (2005), "Some algorithms for the conditional mean vector and covariance matrix"; JSS 2005.

[625] Moon, H., Lee, J.J., Ahn, H., & Nikolova, R.G. (2002), "Web-based simulator for sample size and power estimation in animal carcinogenicity studies", *JSS*, 2002.

[626] Moré, J.J. (1978), "The Levenberg-Marquardt Algorithm: Implementation and Theory", in *Lecture Notes in Mathematics 630*, ed. G.A. Watson, Springer Verlag, Berlin-Heidelberg-New York, 105-116.

[627] Moré, J.J., Garbow, B.S., & Hillstrom, K.E. (1980), *User Guide for MINPACK-1*, Argonne National Laboratory, Argonne, IL.

[628] Moré, J.J., Garbow, B.S., & Hillstrom, K.E. (1981), "Fortran Subroutines for Testing Unconstrained Optimization Software", *ACM Trans. Math. Software*, **7**, 17-41.

[629] Moré, J.J. & Sorensen, D.C. (1983), "Computing a Trust-Region Step", *SIAM Journal on Scientific and Statistical Computation*, **4**, 553-572.

[630] Moré, J.J. and Wright, S.J. (1993), *Optimization Software Guide*, Philadelphia: SIAM.

[631] Morgan, B.J.T. (1992), *Analysis of Quantal Response Data,* London: Chapman & Hall.

[632] Mosteller, F. & Tukey, J.W. (1977), *Data Analysis and Regression: A Second Course in Statistics*, Reading: Addison-Wesley.

[633] Möttönen, J. & Oja H. (1995), "Multivariate spatial sign and rank methods"; Journal of Noparametric Statistics, **5**, 201-213.

[634] Mudholkar, G. S., McDermott, M., & Srivastava, D. K. (1992), "A test of p-variate normality", *Biomerika*, **79**, 850-854.

[635] Mulaik, S.A. (1972) *The Foundations of Factor Analysis*, New York: McGraw Hill Comp.

[636] Mulaik, S.A., James, L.R., Van Alstine, J., Bennett, N., Lind, S., & Stilwell, C.D. (1989), "Evaluation of Goodness-of-Fit Indices for Structural Equation Models", *Psychological Bulletin*, **105**, 430-445.

[637] Murtagh, B. A. (1981), "Advanced Linear Programming: Computation and Practice", McGraw - Hill, New York.

[638] Murtagh, B.A. & Saunders, M.A. (1983), *MINOS 5.0 User's Guide*; Technical Report SOL 83-20, Stanford University.

[639] Muthèn, B.O. (1987), LISCOMP: *Analysis of Linear Structural Relations Using a Comprehensive Measurement Model*, Mooresville IN: Scientific Software, Inc.

[640] Nakaya, T. (1997), "Statistical inferences in bidimensional regression models", *Geographical Analysis*, **29**, 169-186.

[641] Narula, S.C. & Wellington, J.F. (1977), "Prediction, linear regression, and minimum sum of relative error", *Technometrics*, **19**, 185-190.

[642] Nazareth, J. L. (1987), "Computer Solutions of Linear Programs", Oxford University Press, New York - Oxford.

[643] Nelder, J.A. & Mead, R. (1965), "A Simplex Method for Function Minimization", *Computer Journal*, **7**, 308-313.

[644] Nemhauser, G. L. & Wolsey, L. A. (1988), *Integer and Combinatorial Optimization*, New York: John Wiley & Sons

[645] Neter, J., Wasserman, W. & Kutner, M.H. (1990), *Applied Linear Statistical Models*, 3rd edition, Burr Ridge IL: R.D. Irwin.

[646] Nevalainen, J. & Oja, H. (2006), "SAS/IML macros for multivariate analysis of variance based on spatial signs"; *JSS*, 2004.

[647] Ng, E. & Peyton, B.W. (1993), "Block sparse Cholesky algorithms on advanced uniprocessor computers", *SIAM J. Sci. Comput.*, **14**, 1034-1056.

[648] Nollau, V. & Hahnewald-Busch, A. (1975), *Statistische Analysen; Mathematische Methoden der Planung und Auswertung von Versuchen*, Leipzig: VEB Fachbuchverlag.

[649] Oberdiek, H. (1999), "PDF information and navigation elements with hyperref, pdfTeX, and thumbpdf", Slides.

[650] Ogasarawa, H. (1998), "Standard errors for rotation matrices with an application to the promax solution"; *British Journal of Math. and Statist. Psychology*, **51**, 163-178.

[651] Ogasarawa, H. (1999), "Standard errors for Procrustes solutions"; *Japanese Psychological Research*, **41**, 121-130.

[652] Ogasarawa, H. (1999), "Stadard errors for the direct oblimin solution with Kaiser's normalization"; *Japanese Journal of Psychology*, **70**, 333-338.

[653] Ogasarawa, H. (2000), *ROSEF: Version 2 User's Guide*, Technical Report: Otaru University of Commerce, Otaru, Japan.

[654] Ogasarawa, H. (2000), "Standard errors for the principal component loadings for unstandardized and standardized variables"; *British Journal of Math. and Statist. Psychology*, **53**, 155-174.

[655] Ogasarawa, H. (2002), "Concise formulas for the standard errors of component loading estimates"; *Psychometrika*, **67**, 289-297.

[656] Osborne, M.R. (1985), *Finite Algorithms in Optimization and Data Analysis*, New York: John Wiley & Sons.

[657] Oja, H. & Randles, R.H. (2004), "Multivariate nonparametric tests"; *Statistical Science*, to appear.

[658] Osborne, M.R. (1985), *Finite Algorithms in Optimization and Data Analysis*, New York: J. Wiley.

[659] Outrata, J., Schramm, H., & Zowe, J. (1991), "Bundle Trust Region Methods: Fortran Codes for Nondifferentiable Optimization; User's Guide"; Technical Report No. 269, Mathematisches Institut der Universität Bayreuth, 1991.

[660] Owen, "Tables for computing bivariate normal probabilities", *Ann. Math. Statist.*, **27**, 1075-1090.

[661] Parlett, B.N. (1980), *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, NJ.

[662] Paige,C.C., & Saunders, M.A. (1975), "Solution of Sparse Indefinite Systems of Linear Equations", *SIAM J. Numer. Anal.*, **12**, pp. 617-629.

[663] Paige,C.C., & Saunders, M.A. (1982), "LSQR: An algorithm for sparse linear equations and sparse least squares", *ACM Transactions on Mathematical Software* **8**, pp. 43-71.

[664] Paige,C.C., & Saunders, M.A. (1982), "Algorithm 583, LSQR: Sparse linear equations and least-squares problems", *ACM Transactions on Mathematical Software*, **8**, pp. 195-209.

[665] Pan, W. (2009), "A SAS/IML Macro for Computing Percentage Points of Pearson Distribution"; JSS 457, 2009.

[666] Pape, U. (1980), "Algorithm 562: Shortest Path Lengths", *ACM TOMS*, **6**, 450-455.

[667] Patefield, W.M. (1981), "Algorithm AS 159. An efficient method for generating $r * c$ tables with given row and columns totals", *Applied Statistics*, **30**, 91-97.

[668] Patefield, M. (2000), "Fast and Accurate Calculation of Owen's T Function", *JSS*, 2000.

[669] Pavlov, I. (2010), *Online Documentation for the 7-zip program*, `http://7-zip.org/7z.html`.

[670] *PCTEX32: User Manual*, Personal TEX, Inc., Mill Valley, CA.

[671] Pearson, E.S. & Hartley, H.O. (1972), *Biometrica Tables for Statisticians*, Vol II, New York: Cambridge University Press.

[672] Persson, P.O. & Strang, G. (2004), "A simple mesh generator in Matlab"; *SIAM Review*, **46**, 329-345.

[673] Piessens, R., de Doncker, E., Überhuber, C.W., & Kahane, D.K. (1983), *QUADPACK: A Subroutine Package for Automatic Integration*, Berlin: Springer Verlag.

[674] Pinar, M.C. & Elhedhli, S. (1998), "A penalty continuation method for the l_∞ solution of overdetermined linear systems"; BIT .

[675] Pinar, M.C. & Hartmann, W. (1999), "A Fast Huber Approximation Algorithm for Nonlinear ℓ_1 estimation", Sixth SIAM Conference on Optimization, Atlanta.

[676] Pinar, M.C. & Hartmann, W. (2006), "Huber Approximation Algorithm for Nonlinear ℓ_1 estimation", *European Journal of Operational Research*, **109**, 1096-1107.

[677] Pison, G., Rousseeuw, P.J., Filzmoser, P., & Croux, C. (2003), "Robust Factor Analysis"; *Journal of Multivariate Analysis*, **84**, 2003, 145-172.

[678] Platt, J. (1999), "Sequential Minimal Optimization: A fast algorithm for training support vector machines", in B. Schölkopf, C.J.C. Burges, and A.J. Smola (eds), *Advances in Kernel Methods: Support Vector Learning*, Cambridge: MIT Press.

[679] Pohlabeln, H., Wild, P., Schill, W., Ahrens, W., Jahn, I., Bohn-Audorff, U., Jöckel, K.H. (2002), "Asbestos fibreyears and lung cancer: A two-phase case-control study with expert exposure assessment", *Occupational and Environmental Medicine*, **59**, 410-414.

[680] Polak, E. (1971), *Computational Methods in Optimization*, New York, San Francisco, London: Academic Press, Inc.

[681] Powell, J.M.D. (1970) "A hybrid method for nonlinear equations", in *Numerical Methods for Nonlinear Algebraic Equations*, ed. by Rabinowitz; Gordon and Breach.

[682] Powell, J.M.D. (1977), "Restart Procedures for the Conjugate Gradient Method", *Mathematical Programming*, **12**, 241-254.

[683] Powell, J.M.D. (1978a), "A fast algorithm for nonlinearly constraint optimization calculations", in *Numerical Analysis, Dundee 1977, Lecture Notes in Mathematics 630*, ed. G.A. Watson, Springer Verlag, Berlin, 144-175.

[684] Powell, J.M.D. (1978b), "Algorithms for nonlinear constraints that use Lagrangian functions", *Mathematical Programming*, **14**, 224-248.

[685] Powell, M.J.D. (1982a), "Extensions to subroutine VF02AD", in *Systems Modeling and Optimization, Lecture Notes In Control and Information Sciences 38*, in: R.F. Drenick and F. Kozin (eds.), Springer Verlag, Berlin, 529-538.

[686] Powell, J.M.D. (1982b), "VMCWD: A Fortran subroutine for constrained optimization", *DAMTP 1982/NA4*, Cambridge, England.

[687] Powell, M.J.D. (1988), "A tolerant algorithm for linearly constrained optimization calculations", Report DAMTP/1988/NA17.

[688] Powell, J.M.D. (1992), "A Direct search optimization method that models the objective and constraint functions by linear interpolation", *DAMTP/NA5*, Cambridge, England.

[689] Powell, J.M.D. (2000), "UOBYQA: Unconstrained Optimization By Quadratic Approximation", Report DAMTP 2000/NA14, University of Cambridge.

[690] Powell, J.M.D. (2003), "On the use of quadratic models in unconstrained minimization without derivatives", Report DAMTP 2003/NA03, University of Cambridge.

[691] Powell, J.M.D. (2014), "On fast trust region methods for quadratic models with linear constraints", Report DAMTP 2014/NA02, University of Cambridge.

[692] Pregibon, D. (1981), "Logistic Regression Diagnostic", *Annals of Statistics*, **9**, 705-724.

[693] Prescott, P. (1975), "An approximate test for outliers in linear models", *Technometrics*, **17**, 129-132.

[694] Quinlan, J.R. (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufman: San Mateo, CA.

[695] Rahtz, S. (1998), "Hypertext marks in LaTeX: the hyperref package", `http://www.tug.org`

[696] Ramirez, D. E. (2000), "The Generalized F Distribution", *JSS*, 2000.

[697] Ramirez, D.E. & Jensen, D.R. (1991), "Misspecified T^2 Tests. II Series Expansions", *Commun. Statist. - Simula.* **20**, 97-108.

[698] Ramsay, J. O. (1969), "Some statistical considerations in multidimensional scaling"; *Psychometrika*, **34**, 167-182.

[699] Ramsay, J. O. (1977), "Maximum likelihood estimation in multidimensional scaling"; *Psychometrika*, **42**, 241-266.

[700] Ränner, S., Lindgren, F. Geladi, P. & Wold, S. (1994), "A PLS algorithm for data sets with many variables and few objects", *Journal of Chemometrics*, **8**, 111-125.

[701] Rao, C.R. & Mitra, S.K. (1971), *Generalized Inverse of Matrices and Its Applications*, New York: John Wiley & Sons.

[702] Rasch, G. (1960), *Probabilistic models for some intelligence and attainment tests*; Danmarks Pedagogiske Institut, Copenhagen.

[703] Reed, B. C. (1989), "Linear least-squares fits with errors in both coordinates"; *American Journal of Physics*, **57**, 642-646.

[704] Reed, B. C. (1992), "Linear least-squares fits with errors in both coordinates II: Comments on parameter variances"; *American Journal of Physics*, **61**, 59-62.

[705] Reinelt, G. (1991), "TSPLIB - A Traveling Salesman Problem Library", *ORSA J. Comput.*, **3-4**, 376-385.

[706] Richardson, J.D. (2011), "Nonlinear pseudo random number generation using digit isolation and entropy buffers"; submitted to JSS, 2011.

[707] Rohlf, F. J. (1978), "A probabilistic minimum spanning tree algorithm", *Information Processing Letters*, **7**, 44-48.

[708] Rorabacher, D.B. (1991), "Statistical treatment for rejection of deviant values: Critical values of Dixon Q parameter and related subrange ratios at the 95 percent confidence level", *Analytical Chemistry*, **63**, 139-146.

[709] Rosenbrock, H.H. (1960), "An Automatic Method for Finding the Greatest or Least Value of a Function", *Computer Journal*, **3**, 175-184.

[710] Rosenkrantz, D. J., Stearns, R.E. & Lewis, P. M. (1977), "An analysis of several heuristics for the traveling salesman problem", *SIAM Journal on Computing*, **6**, 563-581.

[711] Rosenstein, M.T., Collins, J. J., De Luca, C. J. (1993), "A practical method for calculating largest Lyapunov exponents from small data sets", *Physica*, D 65, 117.

[712] Rosipal, R. & Trejo, L.J. (2001), "Kernel partial least squares regression in reproducing kernel Hilbert space", *Journal of Machine Learning*, **2**, 97-123.

[713] Rousseeuw, P.J. (1984), "Least Median of Squares Regression", *Journal of the American Statistical Association*, **79**, 871-880.

[714] Rousseeuw, P.J. (1985), "Multivariate Estimation with High Breakdown Point", in *Mathematical Statistics and Applications*, Dordrecht: Reidel Publishing Company, pp. 283-297.

[715] Rousseeuw, P.J. & Croux, C. (1993), "Alternatives to the Median Absolute Deviation", *Journal of the American Statistical Association*, **88**, 1273-1283.

[716] Rousseeuw, P.J. & Hubert, M. (1997), "Recent developments in PROGRESS", in: Y. Dodge (ed.): L_1-*Statistical Procedures and Related Topics*, Institute of Mathematical Statistics, Lecture Notes, Vol. 31.

[717] Rousseeuw, P.J. & Leroy, A.M. (1987), *Robust Regression and Outlier Detection*, New York: John Wiley & Sons.

[718] Rousseeuw, P.R. & Van Driessen, K. (1997), "A fast Algorithm for the Minimum Covariance Determinant Estimator", paper presented at the Third International Conference on the L_1 Norm and Related Methods, Neuchâtel, Switzerland.

[719] Rousseeuw, P.J. & Van Zomeren, B.C. (1990), "Unmasking Multivariate Outliers and Leverage Points", *Journal of the American Statistical Association*, **85**, 633-639.

[720] Royston, J. P. (1983), "Some techniques for assessing multivariate normality based on the Shapiro-Wilk *W*", *Applied Statistics*, **32**, 121-133.

[721] Royston, J. P. (1992), "Approximating the Shapiro-Wilk W test for non-normality", *Statistics and Computing*, **2**, 117-119.

[722] Rubin, H. & Johnson, B.C. (), "Efficient generation of exponential and normal deviates"; *Journal of Statistical Computation and Simulation*, **76**, 509-518.

[723] Rudd, A. and Rosenberg, B. (1979), "Realistic Portfolio Optimization", *TIMS Studies in Management Sciences*, **11**, 21-46.

[724] Saad, Y. (1996), *Iterative Metods for Sparse Linear Systems*, Boston: PWS Publishing Company.

[725] Sachs, L. (1974), *Angewandte Statistik*, Berlin, Heidelberg, New York: Springer Verlag.

[726] Samejima, F. (1969), "Calibration of latent ability using a response pattern of graded scores"; *Psychometrika Monograph Supplement*, No. **17**.

[727] Sarle, W. S. (1983), "Cubic Cluster Criterion"; *SAS Technical Report A-108*, Cary NC: SAS Institute Inc.

[728] Sarle, W. S. (1994), "TREEDISC Macro", Cary NC: SAS Institute Inc.

[729] Sarle, W. S. (1995), *The STDIZE SAS Macro*, Cary NC: SAS Institute Inc.

[730] Sarle, W.S. (2000), *The Jackboot Macro*, Cary NC: SAS Institute Inc.

[731] *SAS/IML®Software*, (1989), Version 6, First Ed., SAS Institute Inc., Cary, NC.

[732] *SAS/STAT®User's Guide*, (1990), Version 6, Second Printing, SAS Institute Inc., Cary, NC.

[733] *The SAS®System* (2000), Version 8, SAS Institute Inc., Cary, NC.

[734] SAS Enterprise Miner documentations for PROC ASSOC, SEQUENCE, RULEGEN, etc. version 9.1.3 and 4.3: `http://support.sas.com/documentation/onlinedoc/miner/emtmsas913/TW10113.pdf`
`http://support.sas.com/documentation/onlinedoc/miner/em43/assoc.pdf`
`http://support.sas.com/documentation/onlinedoc/miner/em43/rulegen.pdf`
`http://support.sas.com/documentation/onlinedoc/miner/em43/sequence.pdf`
`http://support.sas.com/documentation/onlinedoc/miner/em43/dmvq.pdf`
`http://support.sas.com/documentation/onlinedoc/miner/em43/neural.pdf`
`http://support.sas.com/documentation/onlinedoc/miner/em43/split.pdf`

[735] SAS Institute Inc. (1990), *SAS Language: Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

[736] *SAS Procedures Guide*, (1990), Version 6, Third Ed., SAS Institute Inc., Cary, NC.

[737] SAS Institute Inc. (1995), SAS Technical Report P-250, *SAS/IML®Software: Changes and Enhancements, through Release 6.11;* SAS Institute Inc., Cary, N.C.

[738] Sasieni, M., Yaspan, A., & Friedman, L. (1968), *Methoden und Probleme der Unternehmensforschung*, Berlin: Verlag Die Wirtschaft

[739] Satorra, A. & Bentler, P.M. (1994), "Corrections to test statistics and standard errors in covariance structure analysis", in: *Latent Variables Analysis*, A. von Eye & C. C. Clogg (ed.), Thousand Oaks: Sage Publications.

[740] Satterthwaite, F. W. (1946), "An approximate distribution of estimates of variance components"; *Biometrics Bulletin*, **2**, 110-114,

[741] Schill, W., Enders, D., & Drescher, K. (2013), "sas-twophase-package: A SAS Package for logistic two-phase studies", paper and software submitted to JSS. Software can be downloaded from: `www.bips.uni-bremen.de/sastwophase`

[742] Schittkowski, K. (1978), "An adaptive precision method for the numerical solution of constrained optimization problems applied to a time-optimal heating process", in *Proceedings of the 8th IFIP Conference on Optimization Techniques*, Springer Verlag, Heidelberg, New York.

[743] Schill, W., Wild, P., & Pigeot, I. (2007), "A planning tool for two-phase case-control studies", *Computer Programs and Methods in Biomedicine*, **88**, 175-181.

[744] Schittkowski, K. (1987), *More Test Examples for Nonlinear Programming Codes*, *Lecture Notes in Economics and Mathematical Systems 282*, Springer Verlag, Berlin-Heidelberg-New York.

[745] Schittkowski, K. & Stoer, J. (1979), "A Factorization Method for the Solution of Constrained Linear Least Squares Problems Allowing Subsequent Data Changes", *Numer. Math.*, **31**, 431-463.

[746] Schmid, J. & Leiman, J.M. (1957), "The development of hierarchical factor solutions", *Psychometrika*, **22**, 53-61.

[747] Schnabel, R.B. & Eskow, E.A. (1990), *SIAM Journal on Scientific and Statistical Computation*, **11**, 1136-1158.

[748] Schölkopf, B. (2000), "Statistical Learning and Kernel Methods", Technical Report MSR-TR-2000-23, Microsoft Research Limited, Cambridge UK.

[749] Schoenemann, P. H. (1972), "An algebraic solution for a class of subjective metrics models"; *Psychometrika*, **37**, 441-451.

[750] Schrage, L. (1979), "A more portable random number generator", ACM TOMS **5** 132-138.

[751] Schreiber, T. and Schmitz, A. (1996), "Improved Surrogate Data for Nonlinearity Tests", *Physical Review Letters*, **77**, 635-638.

[752] Schrepp, M. (1999), "On the empirical construction of implications on bi-valued test items"; *Mathematical Social Sciences*, **38**, 361-375.

[753] Schrepp, M. (2003), "A method for the analysis of hierarchical dependencies between items of a questionnaire", *Methods of Psychological Research*, **19**, 43-79.

[754] Schrepp, M. (2006), *ITA 2.0: A program for classical and inductive item tree analysis*, JSS 2006.

[755] Searle, S.R. (1971), *Linear Models*, New York: John Wiley & Sons.

[756] Serneels, S., Filzmoser, P., Croux, C., & Van Espen, P.J. (2004), "Robust continuum regression"; Technical Report.

[757] Shaffer, J.P. (1995), "Multiple hypothesis testing", *Annual Review of Psychology*, **46**, 561-576.

[758] Shanaz, F., Berry, M.W., Pauca, V.P. & Plemmons, R.J. (2004), "Document clustering using nonnegative matrix factorization"; *J. of Information Processing and Management*, to appear.

[759] Shao, J. & Tu, D. (1995), *The Jackknife and Bootstrap*, New York: Springer Verlag.

[760] Shapiro, S. S. & Wilk, M. B. (1965), "An analysis of variance test for normality (Complete Samples)", *Biometrika*, **52**, 591-611.

[761] Sharpe, W.F. (1987), "An algorithm for portfolio improvement", in: K.D. Lawrence, J.B. Guerard Jr, and G.R. Reeves (ed.), *Advances in Mathematical Programming and Financial Planning*, London: JAI Press Inc., 155-169.

[762] Shechter, G. (2004), *Matlab package kDtree*

[763] Shepard, R. N. (1962), "Analysis of proximities: Multidimensional scaling with an unknown distance function"; *Psychometrika*, **27**, 125-140, 219-246.

[764] Sheppard, K. (2009), *Oxford MFE Toolbox*, Oxford. `kevin.sheppard@economics.ox.ac.uk`

[765] Silvapulle, M. J. & Sen, P. K. (2005): *Constrained Statistical Inference*, New Jersey: Wiley.

[766] Simard, R. & L'Ecuyer, P. (2010), "Computing the Two-Sided Kolmogorov-Smirnov Distribution", Journal of Statistical Software.

[767] Simonetti, N. (1998), "Subroutines for dynamic program for the Traveling Salesman Problem", `www.contrib.andrew.cmu.edu/ neils/tsp/index.html`

[768] Sison, C.P. & Glaz, J. (1995), "Simultaneous confidence intervals and sample size determination for multinomial proportions", *JASA*, **90**, 366-369.

[769] Skibicki, M. & Wywial, J. (2000), "On optimal sample allocation in strata"; Technical Report, Dept. of Statistics, University of Economics, Katowice.

[770] Small, N (1980), "Marginal skewness and kurtosis in testing multivariate normality", *Applied Statistics*, **29**, 85-87.

[771] Small, N (1985), "Multivariate normality, testing for", in Kotz,, S., Johnson, N.L., & Read, C.B. (eds.), *Encyclopedia of Stastistical Sciences*, **6**, Amsterdam: North Holland.

[772] Smith, B.T., Boyle, J.M., Dongarra, J.J., Garbow, B.S., Ikebe, Y., Klema, V.C. & Moler, C.B. (1976), *Matrix Eigensystem Routines - EISPACK Guide*, *Lecture Notes in Computer Science*, Vol. 6, 2nd ed., Springer Verlag, Berlin.

[773] Sobel, R.A. (1982), "Asymptotic confidence intervals for indirect effects in structural equations", in: *Sociological Methodology*, S. Leinhardt (ed.), Washington, DC: American Sociological Association.

[774] Sobel, R.A. (1986), "Some new results on indirect effects and their standard errors in covariance structure models", in: *Sociological Methodology*, N.B. Tuma (ed.), Washington, DC: American Sociological Association.

[775] Somerville, P.N. (1997), "Multiple testing and simultaneous confidence intervals: calculation of constants", *Computational Statistics and Data Analysis*, **25**, 217-233.

[776] Somerville, P.N. (1998), "Numerical computation of multivariate normal and multivariate-t probabilities over convex regions", *Journal of Computational and Graphical Statistics*.

[777] Somerville, P.N. & Bretz, F. (2001), "FORTRAN 90 and SAS-IML programs for computation of critical values for multiple testing and simultaneous confidence intervals", *Journal of Statistical Software*.

[778] Spaeth, H. (1987), *Mathematische Software zur Linear Regression*, München: R. Oldenbourg Verlag GmbH.

[779] Srivastava, J. (1975), "Designs for searching non-negligible effects", in: *A Survey of Statistical Design and Linear Models*, 507-519. Amsterdam: Elsevier, North Holland.

[780] Stadlober, E. (1989), "Ratio of uniforms as a convenient method for sampling from classical discrete distribution", *Proceedings of the 21st conference on Winter simulation*.

[781] Stadlober, E. & Zechner, H. (1999), "The patchwork rejection technique for sampling from unimodal distributions", *ACM Transactions on Modeling and Computer Simulation*, **9**, 59-80.

[782] Stefanski, L.A. & Cook, J. R. (1994), "Simulation-Extrapolation: The Measurement Error Jackknife", JASA **90**, 1247-1256.

[783] Spelucci, P., & Hartmann, W. (1999), "A QR decomposition for matrix pencils", *BIT*, **40**, 183-189.

[784] Steiger, J.H. & Lind, J.C. (1980), "Statistically Based Tests for the Number of Common Factors", paper presented at the annual meeting of the Psychometric Society, Iowa City, IA.

[785] Stiefel, E.L. (1963), *An Introduction to Numerical Mathematics*, New York: Academic Press.

[786] Stine, R. (1989), "An introduction to the bootstrap methods: Examples and ideas", *Sociological Methods and Research*, **18**, 243-291.

[787] Stoppiglia, H., Dreyfus, G. Dubois, R.& Oussar, Y. (2003), "Ranking a random feature for variable and feature selection", *Journal of Machine Learning Research*, **3**, 1399-1414.

[788] Stromberg, A.J. (1993), "Computation of high breakdown nonlinear regression parameters", *JASA*, **88**, 237-244.

[789] Suykens, J.A.K. & Vandevalle, J. (1999), "Least squares support vector classifiers", *Neural Processing Letters*, **9**, 293-300.

[790] Suykens, J.A.K., Lukas, L., Van Dooren, P., De Moor, B., & Vandewalle J. (1999), "Least squares support vector machine classifiers : a large scale algorithm", in *Proc. of the European Conference on Circuit Theory and Design (ECCTD'99)*, Stresa, Italy; 839-842. Technical Report, Kath. University of Leeuven.

[791] Svetnik, V., Liaw, A., Tong, C. & Wang, T. (2004), "Application of Breiman's random forest to modeling structure-activity relationships of pharmaceutical molecules"; in F. Roll, J. Kittler, & T. Windeatt (eds.): *MCS 2004*, LNCS 3077, 334-343.

[792] Swaminathan, H. (1974), "A General Factor Model for the Description of Change", Report LR-74-9, Laboratory of Psychometric and Evaluative Research, University of Massachusetts.

[793] Swarztrauber, P.N. (1982), "Vectorizing the FFT's", in Rodriguez (ed.) *Parallel Computations*, 51-83, New York: Academic Press.

[794] Szekely, G. J. & Rizzo, M. (2005), "A new test for multivariate normality," *Journal of Multivariate Analysis*, **93**, 58-80.

[795] Tagliasacchi, A. (2008), *Matlab package kDtree*

[796] Takane, Y. (1977), "On the relations among four methods of multidimensional scaling"; *Behaviometrika*, **4**,29-42.

[797] Takane, Y., Young, F. W. & de Leeuw, J. (1977), "Nonmetric individual differences multidimensional scaling: An alternating least squares method with optimal scaling features"; *Psychometrika*, **42**, 7-67.

[798] Takane, Y., Young, F. W. & de Leeuw, J. (1980), "An individual differences additive model: An alternating least squares method with optimal scaling features"; *Psychometrika*, **45**, 183-209.

[799] Talebi, H. & Esmailzadeh, N. (2011a), "Using Kullback-Leibler distance for performance evaluation of search designs", *Bulletin of the Iranian Mathematical Society*, **37**, 269-279.

[800] Talebi, H. & Esmailzadeh, N. (2011b), "Weighted searching probability for classes of equivalent search design comparison", *Communication in Statistics: Theory and Methods*, **40**, 635-647.

[801] Tarjan, R.E. (1972), "Depth first search and linear graph algorithms", *SIAM Journal on Computing*, **1**, 146-160.

[802] Thayananthan, A. (2005) "Template based pose estimation and tracking of 3D hand motion", PhD Thesis, Dept. of Engineering, University of Cambridge.

[803] Theiler, J., Eubank, S., Longtin, A., Galdrikian, B. (1992a), "Testing for Nonlinearity in Time Series: the Method of Surrogate Data", *Physica D*, **58**, 77-94.

[804] Theiler, J. et al (1992b), "Using Surrogate Data to Detect Nonlinearity in Time Series"; in "Nonlinear Modeling and Forecasting", edt. Casdagli, M. & Eubank, S., Addison-Wesley, Reading, MA, 163-188.

[805] Therneau, T.M. & Grambsch, P.M. (2000), *Modeling Survival Data, Extending the Cox Model*, New York, Springer.

[806] Thissen, D. & Steinberg, L. (1986), "A taxonomy of item response models"; *Psychometrika*, **51**, 567-577.

[807] Thompson, R. (1985), "A note on restricted maximum likelihood estimation with an alternative outlier model"; *Journal of the Royal Statistical Society*, Ser. B, **47**, 53-55.

[808] Thurstone, L.L. (1931), "Multiple factor analysis"; *Psych. Rev.*, **38**, 406-427.

[809] Tibshirani, R. (1996), "Regression shrinkage and selection via the Lasso", *J. Royal Stat. Soc., Ser. B*, **58**, 267-288.

[810] Tipping, M. E. (2001), "Sparse Bayesian learning and the relevance vector machine", *The Journal of Machine Learning Research*, **1**, 211-244.

[811] Tobler, W.R. (1965), "Computation of the corresponding of geographical patterns", *Papers of the Regional Science Association*, **15**, 131-139.

[812] Tobler, W. R. (1966), "Medieval distortions: Projections of ancient maps", *Annals of the Association of American Geographers*, **56**, 351-360.

[813] Tobler, W. R. (1994), "Bidimensional regression", *Geographical Analysis*, **26**,187-212.

[814] Tomizawa, S. (1987), "Decomposition for 2-ratio-parameter symmetry model in square contingency tables with ordered categories"; *Biom. Journal*, **1**, 45-55.

[815] Torgerson, W. S. (1958), *Theory and Methods of Scaling*, New York: Wiley.

[816] Trindade, A. A. (2003), "Implementing modified Burg algorithms in multivariate subset autoregressive modeling", *JSS*, 2003.

[817] Trujillo-Ortiz, A., Hernandez-Walls, R., Barba-Rojo, K., & Castro-Perez, A. (2007a), "AnDartest: Anderson-Darling test for assessing normality of a sample data", `http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectID=14807`.

[818] Trujillo-Ortiz, A., Hernandez-Walls, R., Barba-Rojo, K., & Cupul-Magana, L. (2007b), "HXmvntest: Henze-Zirkler's multivariate normality test", `http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectID=17931`.

[819] Tucker, L.R. & Lewis, C. (1973), "A reliability coefficient for maximum likelihood factor analysis", *Psychometrika*, **38**, 1-10.

[820] Tukey, J.W. (1977a), *Data Analysis and Regression*, Reading: Addison-Wesley.

[821] Tukey, J.W. (1977b), *Exploratory Data Analysis*, Reading: Addison-Wesley.

[822] Tyler, D.E. (1987), "A distribution-free M-estimator of multivariate scatter", *The Annals of Statistics*, **15**, 234-251.

[823] Van der Voet, H. (1994), "Comparing the predictive accuracy of models using a simple randomization test", *Chemometrics and Intelligent Laboratory Systems*, **25**, 313-323.

[824] van der Vorst, H. (2000), *Iterative Methods for Large Linear Systems*, Utrecht: Utrecht University.

[825] Van Gestel T., Suykens J., De Brabanter J., De Moor B., & Vandewalle J., (2001), "Kernel Canonical Correlation Analysis and Least Squares Support Vector Machines", in *Proc. of the International Conference on Artificial Neureal Networks (ICANN 2001)*, Vienna, Austria; 381-386.

[826] Van Gestel, T., Suykens, J., Lanckriet, G., Lambrechts, A., De Moor, B., & Vandewalle, J., (2002), "Multiclass LS-SVMs : Moderated outputs and coding-decoding schemes", *Neural Processing Letters*, **15**, 45-48. Technical Report, Kath. University of Leeuven.

[827] Van Huffel, S. & Vandewalle, J. (1991), *The Total Least Squares Problem*, SIAM Philadelphia, PA.

[828] van Leeuwe, J. F. J. (1974), "Item Tree analysis", *Nederlands Tijdschrift voor de Psycholoie*, **29**, 475-484.

[829] Vansina F. & De Greve, J.P. (1982), "Close binary systems before and after mass transfer", Astrophys. Space Sci., **87**, 377-401.

[830] Vapnik, V.N. (1995), *The Nature of Statistical Learning*, New York: Springer.

[831] Venables, W. N., & Ripley, B. D. (1994), "Modern Applied Statistics with S-Plus"; New York: Springer.

[832] Venzon, D.J. & Moolgavkar, S.H. (1988), "A Method for Computing Profile-Likelihood-Based Confidence Intervals", *Applied Statistics*, **37**, 87-94.

[833] Vesanto, J., Himberg, J., Alhoniemi, E., & Parhankangas, J. (2000), "SOM Toolbox for Matlab 5"; Technical Report A57, Helsinki University of Technology,.

[834] Volgenant, A. & van den Hout, W. B. (1990), "TSP1 and TSP2 - Symmetric traveling salesman problem for personal computers", Technical Report with Borland Pascal implementation.

[835] Wahba, G. (1990), *Spline Models for Observational Data*, Series in Applied Mathematics, Vol. 59, SIAM Philadelphia.

[836] Wang, C.C. (2011), "TMVN: A Matlab package for multivariate normality test", JSS 2011.

[837] Wang, C.C. & Hwang, Y.T. (2011), "A new functional statistic for multivariate normality", *Statistics and Computing*, **21**, 501-509.

[838] Weber, E. (1972), *Grundriss der biologischen Statistik*, Jena: VEB Gustav Fischer Verlag, 1972.

[839] Weber, E. (1974), *Einführung in die Faktorenanalyse*, Jena: VEB Gustav Fischer Verlag, 1974.

[840] Wedin, P.A. & Lindström, P. (1987), *Methods and Software for Nonlinear Least Squares Problems*, University of Umea, Report No. UMINF 133.87.

[841] Wehrens, R. & Buydens, L. M. C. (2007), "Self and super-organizing maps in R: The Kohonen package"; *JSS*, **21**, 2007.

[842] Weisberg, S. (1980), *Applied Linear Regression*, New York: John Wiley & Sons.

[843] Weisberg, S. (2002), "Dimension reduction regression with R", *JSS*, **7**, 2002.

[844] Weiss, A.A. (1984), "ARMA Models with ARCH Errors," *Journal of Time Series Analysis*, **5**, 129-143.

[845] Welch, B. L. (1947), "The generalization of "Student's" problem when several different population variances are involved"; *Biometrika*, **34**, 28-35

[846] Welch, B. L. (1951), "On the comparison of several mean values: an alternative approach"; *Biometrika*, **38**, 330-336.

[847] Welch, P. D. (1967), "The use of fast Fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms"; *IEEE Transactions on Audio Electroacoustics*, **AU-15(6)**, 70-73.

[848] Weston, J., Mukherje, S., Chapelle, O., Pontil, M., Poggio, T., & Vapnik, V. (2000), "Feature Selection for SVMs", *Neural Information Processing Systems*, **13**, 668-674.

[849] Wheaton, B., Muthèn, B., Alwin, D.F., & Summers, G.F. (1977), "Assessing Reliability and Stability in Panel Models", in *Sociological Methodology*, ed. D.R. Heise, San Francisco: Jossey Bass.

[850] White, H. (2000), "A reality check for data snooping"; *Econometrica*, **68**, 1097-1126.

[851] Wiley, D.E. (1973), "The Identification Problem for Structural Equation Models with Unmeasured Variables", in *Structural Equation Models in the Social Sciences*, eds. A.S. Goldberger and O.D. Duncan, New York: Academic Press.

[852] Wilkinson, J.H. (1963), *Rounding Errors in Algebraic Processes*, Prentice Hall, Englewood Cliffs, NJ.

[853] Wilkinson, J.H. (1965), *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford.

[854] Wilkinson, J.H. & Reinsch, C. (1971), *Handbook for Automatic Computation*, Springer Verlag, Heidelberg.

[855] Willems, G., Pison, G., Rousseeuw, P.J. & Van Aelst, S. (2001), "A robust Hotelling test", Technical Report, University of Antwerp, `http://win-www.uia.ac.be/u/statis`.

[856] Williams, E. (2014), "Aviation Formulary V1.46", `http://williams.best.vwh.net/avform.htm`

[857] Wilson, E.B. & Hilferty, M.M. (1931), "The Distribution of Chi-square", *Proc. Nat. Acad. Sci.*, **17**, 694.

[858] Wold, S. (1994), "PLS for multivariate linear modeling", in: *QSAR: Chemometric Methods in Molecular Design. Methods and Principles in Medicinal Chemistry*, ed. H. van de Waterbeemd, Weinheim: Verlag Chemie.

[859] Wold, S. (1996), "Estimation of principal components and related models by iterative least squares", in: *Multivariate Analysis*, ed. P.R. Krisjhnaiah, New York: Academic Press, 391-420.

[860] Wright, S. J. (1997), *Primal-Dual Interior Point Methods*, SIAM, Philadelphia.

[861] Wright, S. P. (1992), "Adjusted p-values for simultaneous inference", *Biometrics*, **48**, 1005-1013.

[862] Xie, X-J., Pendergast, J., & Clarke, W. (2008), "Increasing the power: A practical approach to goodness-of-fit test for logistic regression models with continuous predictors"; *Computational Statistics and Data Analysis*, **52**, 2703-2713.

[863] Xie, X-J. & Bian, A (2009), "A SAS Package for evaluating logistic and proportional odds model fit"; Submission to *JSS*.

[864] Yang, J. & Honavar, V. (1997), "Feature selection using a genetic algorithm", Technical Report, Iowa State University.

[865] Yates, F. (1960), *Sampling Methods Censuses and Surveys*, London: Griffin & Company Lth.

[866] Young, F. W. (1970), "Nonmetric multidimensional scaling: Recovery of metric information"; *Psychometrika*, **35**, 455-473.

[867] Young, F. W. (1975), "Methods of describing ordinal data with cardinal models", *Journal of Mathematical Psychology*, **12**, 416-436.

[868] Young, F. W. (1987), "Multidimensional scaling: History, Theory, and Applications"; Hillsdale NJ: Lawrence Erlbaum.

[869] Yuan, K.-H., Guarnaccia, C.A. & Hayslip Jr., B. (2003), "A study of the distribution of sample coefficient alpha with the Hopkins symptom checklist: Bootstrap versus asymptotics"; *Educational and Psychological Measurement*, **63**, 5-23.

[870] Zamar, D., Graham, J., & McNency, B. (2007), "elrm: Software implementing exact-like inference for logistic regression models"; *JSS*, **21**.

[871] Zamar, D., Graham, J., & McNency, B. (2013), "Package `elrm`", in CRAN.

[872] Zhang, H.H. (2006), "Variable selection for support vector machines via smoothing spline ANOVA," *Statistica Sinica*, **16(2)**, 659-674.

[873] Zhang, H.H., Ahn, J., Lin, X., & Park, C. (2006), "Gene selection using support vector machines with nonconvex penalty," *bioinformatics*, **22**, pp. 88-95.

[874] Zhang, X., Lobeiza, F.R., Zhang, M.J. & Klein, J.P. (2007), "A SAS Macro for estimation of direct adjusted survival curves based on a stratified Cox regression model"; *Computer Methods and Programme in Biomedicine*, **88**, 95-101.

[875] Zhu, C., Byrd, R.H., Lu, P., & Nocedal, J. (1994) "L-BFGS-B: FORTRAN Subroutines for Large Scale Bound Constrained Optimization", Tech. Report, NAM-11, EECS Department, Northwestern University, 1994.

[876] Zhu, J., Rosset, S., Hastie, T., & Tibshirani, R. (2003), "1-norm support vector machines," *Neural Inform. Processing Systems*, **16**, pp. 49-56.

[877] Ziff, R. M. (1998), "Four-tap shift-register-sequence random-number generators", *Computers on Physics*, **12**, 385-392.

[878] Zou, H., Hastie, T., & Tibshirani, R. (2004), "Sparse principal component analysis"; Technical Report, Stanford University.

Chapter 8

Index

Index