

# Matvec Users' Guide

---

Version 1.03

**Tianlin Wang**

Colorado Springs, Colorado

**Rohan L. Fernando**

Ames, Iowa

**Stephen D. Kachman**

Lincoln, NE

# Acknowledgements

Without generous supports and thoughtful ideas from my academic adviser Prof. Rohan Fernando, there would not have been such a package.

Dr. Ignacy Misztal and I used to have several nice discussions about Matvec development. In particular, his encouragement is greatly appreciated.



# Contents

<b>1</b>	<b>A Tour of Matvec</b>	<b>1</b>
1.1	Running Matvec . . . . .	1
1.1.1	Interactively . . . . .	1
1.1.2	From a script . . . . .	1
1.1.3	From standard input . . . . .	1
1.2	Special Attention . . . . .	1
1.3	Start-up File . . . . .	2
1.4	Creating Objects . . . . .	2
1.5	Getting Online Help . . . . .	3
1.6	Scalar Arithmetic . . . . .	4
1.7	Matrix-Vector Arithmetic . . . . .	4
1.8	Variables, Expressions, and Statements . . . . .	5
1.9	Inbreeding Coefficient Computation . . . . .	6
<b>2</b>	<b>Working on Objects</b>	<b>9</b>
2.1	Special Object: this . . . . .	9
2.2	Scalar Object . . . . .	10
2.3	Vector Object . . . . .	10
2.4	Matrix Object . . . . .	11
2.5	String Object . . . . .	11
2.6	Pedigree Object . . . . .	12
2.6.1	sample . . . . .	12
2.6.2	input . . . . .	12
2.6.3	inbcoef . . . . .	13
2.6.4	logdet . . . . .	13
2.6.5	rela . . . . .	13
2.6.6	reld . . . . .	13
2.6.7	inv . . . . .	13
2.6.8	size . . . . .	13
2.6.9	nbase . . . . .	13
2.6.10	save . . . . .	13
2.7	Data Object . . . . .	13
2.8	Model Object . . . . .	13
2.9	StatDist Object . . . . .	13
<b>3</b>	<b>Matrix Object</b>	<b>15</b>
3.1	Creation . . . . .	15
3.2	Operator . . . . .	16
3.2.1	Arithmetic operators . . . . .	16
3.2.2	Relational operators . . . . .	17
3.2.3	Logical operators . . . . .	17
3.3	Manipulation . . . . .	17

3.3.1	Accessing element and sub-matrix . . . . .	17
3.3.2	Adjoining and stacking . . . . .	18
3.3.3	Selecting elements . . . . .	19
3.4	Computation . . . . .	19
3.4.1	Friend function . . . . .	19
3.4.2	Member function . . . . .	19
3.5	Special Matrices . . . . .	32
3.5.1	ones . . . . .	32
3.5.2	zeros . . . . .	32
3.5.3	identity . . . . .	32
3.5.4	sample . . . . .	33
3.5.5	other special matrices . . . . .	33
<b>4</b>	<b>Program Flow Control</b>	<b>35</b>
4.1	If-Endif and If-Else Statements . . . . .	35
4.2	For Statement . . . . .	36
4.3	While Statement . . . . .	37
4.4	Repeat-Until Statement . . . . .	37
<b>5</b>	<b>File Stream Control</b>	<b>39</b>
5.1	Standard File Streams . . . . .	39
5.1.1	cout . . . . .	39
5.1.2	cin . . . . .	39
5.1.3	cerr . . . . .	39
5.2	User Specified File Stream . . . . .	39
5.2.1	Creation . . . . .	39
5.2.2	Example . . . . .	39
5.3	Member Function . . . . .	40
5.3.1	open . . . . .	40
5.3.2	reopen . . . . .	40
5.3.3	close . . . . .	40
5.3.4	set . . . . .	40
5.3.5	out . . . . .	40
5.3.6	in . . . . .	40
5.3.7	getline . . . . .	40
5.3.8	eof . . . . .	40
5.3.9	endl . . . . .	40
5.3.10	flush . . . . .	40
5.3.11	rewind . . . . .	40
<b>6</b>	<b>Time Control</b>	<b>41</b>
6.1	Digital Clock . . . . .	41
6.1.1	Time . . . . .	41
6.1.2	Localtime . . . . .	41
6.1.3	Gmtime . . . . .	42
6.1.4	Asctime . . . . .	42
6.1.5	Difftime . . . . .	42
6.1.6	Mktime . . . . .	42
6.1.7	Date . . . . .	42
6.1.8	Ctime . . . . .	42
6.1.9	Clock . . . . .	42
6.2	Western Calendar . . . . .	43
6.2.1	Leap year . . . . .	43

<b>7</b>	<b>User-Defined Function</b>	<b>45</b>
7.1	Syntax . . . . .	45
7.2	Description . . . . .	45
7.3	Scope . . . . .	45
7.4	Recursion . . . . .	45
7.5	Examples . . . . .	45
<b>8</b>	<b>2D/3D Object-Oriented Plotting</b>	<b>47</b>
8.1	Creation . . . . .	47
8.2	Member Function . . . . .	47
8.2.1	plot . . . . .	47
8.2.2	plot3D . . . . .	47
8.2.3	replot . . . . .	47
8.2.4	set . . . . .	47
8.2.5	save . . . . .	48
8.2.6	open . . . . .	49
8.2.7	close . . . . .	49
8.3	Examples . . . . .	49
8.3.1	Example 1 . . . . .	49
8.3.2	Example 2 . . . . .	49
8.3.3	Example 3 . . . . .	50
8.3.4	Example 4 . . . . .	50
<b>9</b>	<b>Macro Packages</b>	<b>51</b>
9.1	Demo Package . . . . .	51
9.1.1	demo() function . . . . .	52
9.1.2	demo_src() function . . . . .	53
9.2	Prime Number Package . . . . .	54
9.2.1	prime(n) . . . . .	54
9.2.2	prime_next(n) . . . . .	54
9.2.3	prime_less(n) . . . . .	54
9.2.4	examples . . . . .	54
9.3	Special Matrix Package . . . . .	55
9.3.1	hilb . . . . .	55
9.3.2	invhilb . . . . .	55
9.3.3	hankel . . . . .	55
9.3.4	vander . . . . .	56
9.3.5	hadamard . . . . .	56
9.3.6	pascal . . . . .	56
9.3.7	toeplitz . . . . .	56
9.4	Linear Programming Package . . . . .	57
<b>10</b>	<b>Linear Model Analyses</b>	<b>59</b>
10.1	Linear Regression . . . . .	59
10.2	Best Linear Unbiased Estimation (BLUE) . . . . .	59
10.3	Best Linear Unbiased Prediction (BLUP) . . . . .	59
10.3.1	Data . . . . .	59
10.3.2	Model . . . . .	60
10.3.3	Obtain BLUP using matrix algebra . . . . .	60
10.3.4	Obtain BLUP using Matvec higher level functions . . . . .	61
10.3.5	Multi-trait and multi-model example . . . . .	63
10.4	Linear Estimation . . . . .	64
10.5	Linear Hypothesis Test . . . . .	64
10.6	Least Squares Means (lsmeans) . . . . .	65

10.7 Variance Components Estimation (VCE) . . . . .	66
<b>11 Generalized Linear Mixed Model Analyses</b>	<b>69</b>
11.1 Generalized Linear Model . . . . .	69
11.1.1 Model Specification . . . . .	70
11.1.2 Maximum Likelihood Estimates . . . . .	71
11.1.3 Estimation and Testing . . . . .	71
11.1.4 Threshold Model . . . . .	73
11.1.5 Survival analysis . . . . .	74
11.1.6 Growth Curves . . . . .	77
11.2 Penalized Quasi-Likelihood . . . . .	79
11.2.1 Correlated random effects . . . . .	80
11.2.2 Multivariate Correlated Random Effects . . . . .	81
11.2.3 Cubic splines . . . . .	84
<b>12 Segregation and Linkage Analyses</b>	<b>87</b>
12.1 Genotype Probability Computation . . . . .	87
12.2 Genetic Mapping . . . . .	88
<b>13 Statistical Distributions</b>	<b>89</b>
13.1 Introduction . . . . .	89
13.2 Continuous Distribution . . . . .	90
13.2.1 Normal distribution . . . . .	90
13.2.2 Uniform distribution . . . . .	92
13.2.3 $\chi^2$ distribution . . . . .	93
13.2.4 $t$ distribution . . . . .	95
13.2.5 $F$ distribution . . . . .	96
13.2.6 Gamma distribution . . . . .	98
13.2.7 Exponential distribution . . . . .	99
13.2.8 Beta distribution . . . . .	100
13.2.9 Log normal distribution . . . . .	101
13.3 Discrete Distribution . . . . .	103
13.3.1 Binomial distribution . . . . .	103
13.3.2 Poisson distribution . . . . .	104
13.3.3 Geometric distribution . . . . .	105
13.3.4 Negative binomial distribution . . . . .	107
<b>14 Exception</b>	<b>109</b>
14.1 Error Handling . . . . .	109
14.2 Troubleshooting . . . . .	109
14.2.1 Lack of memory . . . . .	109
14.2.2 Intermediate files . . . . .	109
14.2.3 Bugs . . . . .	109
14.2.4 EPSILON . . . . .	109
14.3 Limitation . . . . .	110
<b>15 Matvec C++ API</b>	<b>111</b>
15.1 An Overview of C++ Class Library . . . . .	111
15.2 Examples . . . . .	111
<b>Index</b>	<b>113</b>

# Preface

The object-oriented problem solving and object-oriented programming represent a way of thinking and a methodology for computer programming that are quite different from the traditional approaches supported by structured programming languages. The powerful features of object-oriented programming support the concepts that make computer problem solving a more human-like activity and that increase the re-usability of software code.

C++ provides programmers and problem solvers object-oriented capability without loss of run-time or memory efficiency. In addition, C++ is available in almost every computer systems from PC to mainframe.

Matvec, written in C++, was particularly designed for animal breeders. There are, however, lots of general operations and functions for matrix-vector. Matvec is an object-oriented, interactive, and interpreted programming language. It provides a comprehensive computing environment where you can solve problems ranging from matrix-vector operation to variance components estimation. It is a powerful tool for teaching and research.

Matvec can't handle complex numbers. This may be a disadvantage for some users. I assume, however, most of Matvec users, as animal breeders, are not interested in complex numbers.



# Chapter 1

## A Tour of Matvec

### 1.1 Running Matvec

#### 1.1.1 Interactively

On most computer systems, Matvec can be invoked on your terminal by entering the shell command `matvec`. Then Matvec will start with an initial message similar to:

```
matvec 0.01, University of Illinois
type ? (or xman or help or xhelp) for on-line help
```

and then a prompt `>` next to the cursor. Now Matvec is ready for you to try something out. For instance, type `a=[1,2,3]`.

#### 1.1.2 From a script

If you want to accomplish a little bit complicated job, then it is a better idea to write a script file using Matvec language. Suppose your script file named `try.mv` contains the following Matvec statements

```
A = [3,2,1; 4,6,5; 7,8,9];
A.det()
```

Then you can type `matvec try.mv` at your computer system prompt to get the determinant of matrix A. You can also redirect the output from Matvec to a file using UNIX redirect mechanism. For instance, under C-shell, the command

```
matvec try.me >try.out&
```

will create a file `try.out` which contains standard output from Matvec. The last character `&` tells the computer to run Matvec in background.

#### 1.1.3 From standard input

Try the following command at your computer system prompt:

```
cat try.mv |matvec
```

### 1.2 Special Attention

There are a few of things to which you have to pay a special attention:

- anything after `#` or `//` to the end of a line is treated as comments, thus ignored by Matvec interpreter. For multi-line comments, use `/* ... */`

- an empty line is ignored by Matvec interpreter.
- backslash \ at the end of a line is a continuation operator.
- the best way to stop Matvec is command **quit**. In an emergency, you can use Cntl-c to terminate Matvec.
- If a statement (expression) is ended with a semicolon (;), then returned value from this statement is not displayed on the screen. In other word, semicolon (;) suppresses printing.
- if \$ is the first character in a line, then anything after \$ is interpreted as the shell command and will be sent to the shell to execute. Almost all shell commands can be accessed and executed within Matvec through this way. For instance, \$ls in UNIX displays the contents of the current working directory.

### 1.3 Start-up File

Whenever you starts Matvec, it tries to read the script file \$HOME/.matvecrc where \$HOME is your home directory. This optional file is referred to as a start-up file where you can put any Matvec statements to initialize some global variables and parameters. Here is an example of \$HOME/.matvecrc:

```
PAGER = "less";

// this.parameter("OUTPUT_PRECISION",16);
// this.parameter("OUTPUT_LINE_WIDTH", 132);
```

### 1.4 Creating Objects

#### Example 1:

```
> A = 4
      4
```

It simply creates a scalar object named A.

#### Example 2:

```
> a=1:4

      i = 1      i = 2      i = 3      i = 4
      1         2         3         4
```

It simply creates a vector object named a. There are four elements 1, 2, 3, and 4.

#### Example 3:

```
> A = [1,2,3;4,5,6]

      Col 1      Col 2      Col 3
Row 1      1         2         3
Row 2      4         5         6

> A.mean()

      i = 1      i = 2      i = 3
      2.5       3.5       4.5
```

The first statement creates a matrix object named A with two rows and three columns. The second statement returns a vector containing the means of each column of A.

**Example 4:**

```
> s = "Hello, My name is Tianlin Wang"
Hello, My name is Tianlin Wang
```

It simply creates a string object named `s`.

## 1.5 Getting Online Help

Matvec provides the online help. There are several versions (different formats) of the same help resources: dvi, ps, html, and manpage. Type in the question mark `?` at the prompt, wait for a second, then a list of topics about Matvec functions and statements will be displayed on your screen. Type `?` followed by a topic name (say `sin`) will display details of the topic you chose. For example,

```
> ?max
```

```
Builtin Friend and Member Function: max
```

**Syntax**

```
max(A)
max(A,B)
A.max()
A.max(B)
```

**Description**

- If `A` is a scalar, `A.max()` returns itself.
- If `A` is a vector, `A.max()` returns an element with the largest value.
- If `A` is a matrix, `A.max()` returns a vector whose each element is the largest element of each column of `A`. If `A` is a one-row or one-column matrix, then the returned value would be a scalar.
- If both `A` and `B` are scalars `A.max(B)` returns the largest one between `A` and `B`.
- If both `A` and `B` are vectors (one of them could be a scalar), then `A.max(B)` returns a vector of the same size as `A` or `B` with resulting element(`i`) being the largest element among `A(i)` and `B(i)`.
- If both `A` and `B` are matrices (one of them could be a scalar), then `A.max(B)` returns a matrix of the same size as `A` or `B` with resulting element(`i,j`) being the largest element among `A(i,j)` and `B(i,j)`.
- `max(A)` is the same as `A.max()`.
- `max(A,B)` is the same as `A.max(B)`.

**See Also**

```
min, sum, sumsq, mean, variance, Scalar, Vector, Matrix.
```

You can obtain the most powerful online help using `xhelp`. It will bring you the hypertext version of Matvec online help resources. This requires a hypertext browser such as netscape or Mosaic installed on your computer system.

## 1.6 Scalar Arithmetic

Matvec is a super calculator. Almost all of arithmetic operations are available. The symbols `+`, `-`, `*`, `/`, `^` represent addition, subtraction, multiplication, division, and power, respectively. For examples,

```
> (2+3)*2^3-8/4
38
> 0/0
NaN
```

Notice that 1) Matvec handles the precedence of operations in a usual way, and 2) there is no difference between integer and real number. In fact they are internally stored in memory as a double precision number.

## 1.7 Matrix-Vector Arithmetic

The simplest way to create a matrix is to type it in at the Matvec prompt:

```
> A = [1,2,3; 4,5,6; 7,8,9]
```

	Col 1	Col 2	Col 3
Row 1	1	2	3
Row 2	4	5	6
Row 3	7	8	9

Note that the symbol pair `[]` is one of Matvec special operators. Matvec is ready to create a matrix object whenever it recognizes this pair. Everything inside the `[]` is the contents of a matrix. Each element is separated with comma(`,`) and each row is separated with semicolon(`;`). If there is nothing inside of `[]`, then a null (empty) matrix object is created. Any matrix specific operations and functions can be applied to an empty matrix, though most of them may not make sense.

Matvec provides a great number of functions for matrix computation. For example,

```
# let's first create a matrix object named A
> A = [3,2,1; 2,5,3; 1,3,4]
```

	Col 1	Col 2	Col 3
Row 1	3	2	1
Row 2	2	5	3
Row 3	1	3	4

```
# find the largest elements for each columns
> A.max()
```

i = 1	i = 2	i = 3
3	5	4

```
# find the largest element in matrix A
> A.max().max()
```

5

```
# how about the inverse of A, surely, matrix A is non-singular
```

```
> A.inv()
```

	Col 1	Col 2	Col 3
Row 1	0.458333	-0.208333	0.0416667

```

      Row 2    -0.208333    0.458333   -0.291667
      Row 3     0.041667   -0.291667    0.458333

# what about a generalized inverse of matrix A
> A.ginv()

      Col 1      Col 2      Col 3
Row 1    0.458333   -0.208333   0.0416667
Row 2   -0.208333    0.458333   -0.291667
Row 3    0.041667   -0.291667    0.458333

# the eigenvalues from MATVEC is not pre-sorted
> A.eigen()
      i = 1      i = 2      i = 3
2.38677      1.19440      8.41883

```

The singular value decomposition of an m-by-n matrix A is a decomposition

$$A = U*W*V'$$

where U is m-by-n with  $U'*U = I$ , W is an n-by-n diagonal matrix and V is an n-by-n orthogonal matrix.

Both A.svd() and A.svd(U,V) returns a vector containing diagonals in W, but the latter has U and V which must be declared as matrix objects before calling.

```

> A = [3,2,1; 2,5,3; 1,3,4];
> u=[]; v=[];
> A.svd(u,v)
      i = 1      i = 2      i = 3
8.41883      2.38677      1.19440

> u
u =

      Col 1      Col 2      Col 3
Row 1   -0.374359    0.815583   -0.441225
Row 2   -0.725619    0.0386051   0.687013
Row 3   -0.577350   -0.577350   -0.577350

> v
v =

      Col 1      Col 2      Col 3
Row 1   -0.374359    0.815583   -0.441225
Row 2   -0.725619    0.0386051   0.687013
Row 3   -0.577350   -0.577350   -0.577350

```

## 1.8 Variables, Expressions, and Statements

An identifier consists of letters, underscores, and digits with a leading letter or underscore. Identifiers are used to name keywords, functions, and variables. Matvec itself like other language has its own reserved keywords. They are listed here: *and*, *andif*, *break*, *continue*, *else*, *for*, *function*, *if*, *for*, *input*, *or*, *protected*, *quit*, *return*, *repeat*, *until* *while*.

When you name a variable, consider the following:

- make the name long enough to mean something to you. There is no limit on the length of variable name in Matvec.

- make the name easy to read by using the underscore character to separate parts of the name
- never use any of Matvec predefined names.

Matvec is case-sensitive in the names of variables, functions, and statements. For example, *Matrix* and *matrix* is not the same, the former is a function name, and latter is a variable name.

A single number can be an integer, a decimal fraction, or a number in scientific (exponential) notation. Note that a single number is represented within Matvec in double-precision floating point format. Here are examples of single numbers:

```
3.14
+3.14
314e-2
314E-2
.314e+1
-(-3.14)
```

The basic structural element of the Matvec language is the expression:

- a single number such as 3.14 is an expression,
- a matrix (or vector) is an expression
- an assignment such as *variable = expression* is an expression
- most of operations and functions result in expressions

Matvec statements includes expression, assignment, *if-endif*, *if-else*, *while*, *repeat-until*, *for*, *return*, *break*, *continue*, etc. A statement is normally terminated with the carriage return. A statement, however, can be continued to the next line with continuation operator \ followed by a carriage return. On the other hand, several statements can be placed on a single line if separated by either comma or semicolons. If a statement is ended by a semicolon, the printing is suppressed. Most of intermediate expression-statements are not expected to print. Thus, statements should always be ended with semicolons except those you want printing.

## 1.9 Inbreeding Coefficient Computation

Suppose we have a pedigree

```
A1  .  .
A2  .  .
A3 A1 A2
A4  . A2
A5 A3 A4
```

which is stored in a ASCII file called `try.ped`. Matvec provides a very friend interface to compute the inbreeding coefficient for each individual or the average inbreeding of a population or the maximum inbreeding coefficient in a population:

```
/home/tianlin/ matvec
matvec v0.10, Mar. 12, 1995, University of Illinois
type ? (or help, xhelp, xman) for on-line help

# the 1st step is to create a Pedigree object, and input ASCII file
> p = Pedigree();
> p.input("try.ped");

# now, compute the inbreeding coefficient for each individual
```

```
> p.inbcoef()

      i = 1      i = 2      i = 3      i = 4      i = 5
      0         0         0         0         0.125
# and the average of inbreeding coefficients for the population
> p.inbcoef().mean()
0.025

# finally the maximum inbreeding coefficient in the population
> p.inbcoef().max()
0.125
```

You may immediately complain about the size of pedigree that Matvec are dealing with. Try a pedigree with a half million individuals, and compute the same things as you did for five-individual pedigree. Don't be surprised if Matvec gives your answer within minutes.





## Chapter 2

# Working on Objects

Matvec is object-oriented. Almost every piece in Matvec system is an object. All objects in Matvec can be classified into nine classes: *Scalar*, *Vector*, *Matrix*, *SparseMatrix*, *String*, *Data*, *Model*, *FileStream*, and *StatDist*.

It should always keep in mind that you're using an object-oriented language. Thus, to play with Matvec you have to create your own objects at the very beginning.

### 2.1 Special Object: *this*

A very special object built in Matvec is *this*. It is the object of the current working session. All of the parameters, properties pertaining to the current working session can be changed through this special object. For C++ programmer, keyword *this* is an old friend

If you simply type *this*, it will display the names of the user-defined variables and functions.

*this.clear(a,b,...)* deletes the named objects from the current Matvec session immediately. The memory associated with the objects are freed. *this.clear()* deletes all objects (but not the builtins') and all intermediate temporary files (but not ones created from the other matvec sessions) from the matvec trash can. The return value is the number of objects that have been successfully deleted.

*this.parameter( "OUTPUT\_PRECISION", k)* sets the output precision for the current Matvec working session to k. The default value is 6. *this.parameter( "OUTPUT\_PRECISION")* returns the output precision in the current Matvec working session. Note that "precision" is defined to be the number of significant digits in output.

*this.parameter( "WARNING", k)* sets the warning flag for the current working session to k, which can be either 1 or 0. The default value is 1. *this.parameter( "WARNING")* returns the value of the warning flag in the current working session.

*this.parameter( "EPSILON", x)* sets the epsilon value for the current Matvec working session to x. The default value is 1.0e-14. Matvec users are strongly discouraged to change the default epsilon value. *this.parameter( "EPSILON")* returns the epsilon value in the current Matvec working session.

*this.parameter( "OUTPUT\_LINE\_WIDTH", k)* sets the line width of output for the current Matvec working session to k. The default value is 80. *this.parameter( "OUTPUT\_LINE\_WIDTH")* returns the line width of output in the current Matvec working session.

*this.parameter( "INPUT\_LINE\_WIDTH", k)* sets the line (record) width for input in the current Matvec working session to k. The default value is 1024. *this.parameter( "INPUT\ \_LINE\ \_WIDTH")* returns the line (record) width for input in the current Matvec working session.

*this.parameter( "MAX\_ERRORS", k)* sets the maximum number of errors allowed for the current Matvec working session to k. The default value 15. *this.parameter( "MAX\_ERRORS")* returns the maximum number of errors allowed for the current Matvec working session.

For example,

```
> this.parameter("OUTPUT_PRECISION")
```

```

        6
> 2.0*asin(1.0)
        3.14159
> this.parameter("OUTPUT_PRECISION",15)
> 2.0*asin(1.0)
        3.14159265358979

```

## 2.2 Scalar Object

A scalar object in Matvec is defined as a floating point number (double-precision). For example 1.5, .3, 3, -3e-2, etc.

The simplest way to create an object of class *Scalar* is to type in a floating point number at Matvec prompt. The standard arithmetic operators such as +, -, \*, /, ^ are working for scalars. The standard mathematical functions are designed as friend as well as member functions for scalars using the C++ math library (except *max*, *min*, *gammaln*, and *gammainc*). They are listed below:

abs	acos	asin	atan	atan2	ceil	cos
cosh	floor	log	log10	max	min	mod
sin	sinh	sqrt	tan	tanh	gammaln	gammainc

For example,

```

> A = 10;
> A.sin().sin()
    -0.517581

```

Details about the ranges and error conditions for these functions can be found in your math library reference manuals.

A scalar  $x$  can be converted to one-element matrix or vector by using `[x]` and `x:x`. For example,

```

> x=2.5
    2.5
> [x]

           Col 1
    Row 1      2.5
> x:x

    i = 1
    2.5

```

## 2.3 Vector Object

A vector object in Matvec is defined as a one-dimensional array with each element being floating point number of double precision. It can be created using the colon operator and be resized at any time for any size using `A.resize(n)` where `n` is the size you want. For example,

```

> A = 1:4

    i = 1      i = 2      i = 3      i = 4
    1          2          3          4

> B = 1:4:2

    i = 1      i = 2
    1          3

```

Another method to create a vector object is to use one of object-creating function `Vector(...)` as shown below:

```
> C = Vector(-5, 5, B)
      i = 1      i = 2      i = 3      i = 4
      -5        5         1         3
```

The  $i$ 'th element of vector  $A$  can be accessed by  $A(i)$ . Note that vector indexing starts from one rather than zero. The usual algebraic operators such as  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $.*$ ,  $./$ ,  $+=$ ,  $-=$ ,  $*=$ ,  $/=$  are available. If both  $A$  and  $B$  are vectors, then  $A*B$  returns an inner product of vectors  $A$  and  $B$ , whereas  $A.*B$  returns a vector with each element being product of corresponding elements in  $A$  and  $B$ . The element-by-element division can be done by either  $A/B$  or  $A./B$ .

The relational operators are  $==$ ,  $!=$ ,  $<$ ,  $>$ ,  $<=$ ,  $>=$ . The logical operators are  $||$  (OR),  $&&$  (AND) and  $!|$ ,  $!&&$ .

There are a lot of member functions for vectors such as `max`, `min`, `mean`, `input`, `mat`, `diag`, `sin`, etc.

A vector ( $v$ ) can be easily converted to any size of matrix by using `v.mat(m,n)` as long as there are enough elements to fill.

## 2.4 Matrix Object

See details in Chapter 3.

## 2.5 String Object

A string object in Matvec is defined as a sequence of characters. It can be created by using double-quotes operator `"`. For example,

```
s = "Hello World";
```

A string object can also be created from a scalar object using the `string()` function. For example,

```
> "tan(10)="+string(tan(10))
      tan(10)=0.648361
```

The operators and member functions for string objects are very limited. Indexing and concatenation works as usual. For example

```
> s = "Hello World";
> s.size()
      11
> t = "Greetings From MATVEC";
> s + t
      Hello WorldGreetings From MATVEC
> s + ", " + t
      Hello World, Greetings From MATVEC
> s([1,7])
      HW
```

Some characters are so special that they cannot be included literally in a string object. Instead, they are represented with escape sequences. An escape sequence is a two-character sequence beginning with a backslash (`\`) followed by a special character. The following is a list of escape sequences implemented in Matvec:

escape sequences	meanings
\\	a literal backslash (\)
\"	a literal double-quote (")
\n	a newline
\t	a horizontal tab
\f	a formfeed
\b	a backspace
\r	a carriage return
\a	an alert
\v	a vertical tab

For examples,

```
> "MATVEC"
      MATVEC
> "\"MAT\tVEC\""
      "MAT      VEC"
```

## 2.6 Pedigree Object

A Pedigree object in Matvec is a object which holds a pedigree.

For example,

```
> ped_file_name = RawData {
  A1      .      .
  A2      .      .
  A3      A1      A2
  A4      .      A2
  A5      A4      A3
}
      .matvec.798672017.0
> # note that .matvec.798672017.0 is the file-name for
> #           the temporary file
> P = Pedigree();
> P.input(ped_file_name);
> P.inbcoef().max()
      0.125
```

### 2.6.1 sample

P.sample(n,ng,s0,d0,imrate,parent,nopo,nofsib,sexratio) generates a arbitrary pedigree of size n with ng generations starting from s0 base sires and d0 base dams. The argument imrate is the immigration rate; argument parent is a switch with 1 meaning no missing parents are allowed; argument nopo is a switch with 1 meaning no mating between parent and its progeny is allowed; argument nofsib is a switch with 1 meaning no mating between full-sibs is allowed; argument sexratio is the sex ratio of new born progeny (male:female). All arguments except the first one are optional with default values: ng=1, s0=1, d0=1, imrate=0.1, parent=1, nopo=1, nofsib=1, sexratio=0.5.

### 2.6.2 input

P.input("<fname>",<column-names>",<pedtype>") inputs pedigree from the ASCII file "<fname>", and "<column-names>" must contain the three key words in any order: individual mother father. If "<column-names>" is omitted, then Matvec uses the default: "individual mother father". The last argument "<pedtype>" is optional. The possible pedtypes available are "raw","standard","group".

For examples, P.input("skip individual skip father skip skip mother") results in the format "individual father mother". You do not have to worry about the trailing columns.

**2.6.3 inbcoef**

P.inbcoef() returns a vector of inbreeding coefficients for all members in the pedigree.

**2.6.4 logdet**

P.logdet() returns the natural logarithm of the additive numerator relationship matrix.

**2.6.5 rela**

P.rela() returns the additive numerator relationship matrix.

**2.6.6 reld**

P.reld() returns the dominant relationship matrix, assuming no no inbred individuals.

**2.6.7 inv**

P.inv() returns the inverse of the additive relationship matrix.

**2.6.8 size**

P.size() returns the total number of individuals in pedigree P.

**2.6.9 nbase**

The number of base members in pedigree P.

**2.6.10 save**

P.save("<fname>") saves a ASCII copy of pedigree

**2.7 Data Object**

A Data object in Matvec is a object which can handle a dataset. Data class is still under development. A data object can be created by `D = Data()`. Then, use its member function *input* to input data from disk into the object.

**2.8 Model Object**

A model object can be created by `M = Model()`. Then, use its member function *equation* to specify the linear model equations. Other characteristics for the model such as random effect, covariate, etc. can be specified through appropriate member functions.

For example,

```
> M = Model();
> M.equation("y = herd animal");
```

**2.9 StatDist Object**

See details in Chapter 13.



## Chapter 3

# Matrix Object

Matrix computation is a complex task. Matvec provides a comprehensive set of functions and tools, which are powerful and easy to learn, to perform the massive matrix manipulation and sophisticated matrix computation. In particular, Matvec can be used to answer *what-if* questions raised by your research colleagues or by yourself.

### 3.1 Creation

There are several ways to create a matrix object. The three common matrix creation methods are listed below:

- type in all elements row-by-row directly from keyboard with the special operator [ and ]. That is starting with [, separating each elements with comma (,), separating each row with semicolon (;), and ending with ]. For example,

```
A = [1.0, 2.0, -2.0; 3.0, 2.0, sqrt(9.0)];
```

- use object-creating friend functions such as Matrix(m,n), ones(m,n), zeros(m,n), and sample(m,n). For example,

```
> A = Matrix(2,3)      # Caution: its contents could be junk
```

	Col 1	Col 2	Col 3
Row 1	3.24485e-308	2.26268e-76	1.28459e-153
Row 2	3.24492e-308	5.82751e+252	3.94257e+180

```
> B = ones(2,3)
```

	Col 1	Col 2	Col 3
Row 1	1	1	1
Row 2	1	1	1

```
> C = zeros(2,3)
```

	Col 1	Col 2	Col 3
Row 1	0	0	0
Row 2	0	0	0

```
> D = sample(2,3)      // generated from Uniform(0.0,1.0)
```

	Col 1	Col 2	Col 3
Row 1	0.865423	0.122624	0.39272
Row 2	0.199465	0.912348	0.192371

- use object-creating member functions. For example, if  $A$  is a matrix,  $A.inv()$  returns the inverse of  $A$ .

## 3.2 Operator

### 3.2.1 Arithmetic operators

$-A$  unary minus (negation)

$+A$  unary plus

$!A$  negation

$A'$  matrix transpose

$++A$  prefix-increment

$--A$  prefix-decrement

$A++$  post-increment

$A--$  post-decrement

$A = B$  assignment

$A + B$  does element-by-element addition

$A - B$  does element-by-element subtraction

$A * B$  does matrix multiplication

$A / B$  does matrix right division

$A @ B$  Kronecker tensor product of  $A$  and  $B$

$A ^ B$  power

$A .+ B$  does element-by-element addition

$A .- B$  does element-by-element subtraction

$A .* B$  does element-by-element multiplication

$A ./ B$  does element-by-element division

$A .^ B$  does element-by-element power

$A += B$  does short hand addition assignment

$A -= B$  does short hand subtraction assignment

$A *= B$  does short hand multiplication assignment

$A /= B$  does short hand right division assignment

$A .+= B$  does short hand element-by-element addition assignment

$A .-= B$  does short hand element-by-element subtraction assignment

$A .*= B$  does short hand element-by-element multiplication assignment

$A ./= B$  does short hand element-by-element division assignment



### 3.2.2 Relational operators

There are six relational operators, they operate between two objects.

`==` equal to

`!=` not equal to

`<` less than

`>` greater than

`<=` less then or equal to

`>=` greater than or equal to

These six operators are based on element-by-element comparison. It returns a matrix with resulting element(i,j) = 1.0 if the relationship is true, otherwise resulting element(i,j) = 0.0

### 3.2.3 Logical operators

There are two sets of logical operators: (`&&`, `||`) and (`.&&`, `.||`).

The `&&` is a short-circuit logical operator AND. `A && B` returns 1 if both A and B are true (B is evaluated only after A is true), otherwise return 0. If A (or B) is a matrix, then `A.all().all()` is implicitly applied. If A (or B) is a vector, then `A.all()` is implicitly applied. Any other types of objects will be determined false.

The `||` is a short-circuit logical operator OR. `A || B` returns 1 if either A and B are true (B is evaluated only after A is false), otherwise return 0. If A (or B) is a matrix, then `A.all().all()` is implicitly applied. If A (or B) is a vector, then `A.all()` is implicitly applied. Any other types of objects will be determined false.

The `.&&` and `.||` are the same as `&&` and `||` except that 1) the former is performed element-by-element, 2) the former is non-short-circuit, thus both operands will be evaluated.

For readability, Matvec creates synonyms *AND* and *OR* for `&&` and `||`, respectively.

## 3.3 Manipulation

Matrix manipulation is a tricky work. Basically, there are three kinds of manipulations: 1) accessing elements and sub-matrices, 2) adjoining and stacking, 3) selecting elements which satisfy whatever condition you specify. They are described below in detail.

### 3.3.1 Accessing element and sub-matrix

- accessing an element  
`A(i,j)` accesses (i,j)'th element with boundary checking
- accessing a row  
`A(i,*)` gets a copy of the i'th row of matrix A.
- accessing a column  
`A(*,j)` gets a copy of the j'th column of matrix A.
- accessing sub-matrix  
`A(i1:i2, j1:j2)` returns a sub-matrix of A with row i1 through i2 and column j1 through j2. `A([i1,i2,i3], [j1,j2,j3])` returns a sub-matrix of A with rows i1, i2, and i3 and columns j1, j2, and j3.

For example,

```
> A = [1,2,3;4,5,6]
```

	Col 1	Col 2	Col 3
Row 1	1	2	3
Row 2	4	5	6

```
> A(2,3)      # returns element at row 2 and column 3
6
```

```
> A(*,2)      # returns the second column
```

	Col 1
Row 1	2
Row 2	5

```
> A(1,*)      # returns the first row
```

	Col 1	Col 2	Col 3
Row 1	1	2	3

```
> A(2,1:3)    # returns the second row
```

	Col 1	Col 2	Col 3
Row 1	4	5	6

```
> A(*,[3,2,1]) # the columns are reversed
```

	Col 1	Col 2	Col 3
Row 1	3	2	1
Row 2	6	5	4

### 3.3.2 Adjoining and stacking

Adjoining or stacking matrices in Matvec are quite convenient. Suppose both A and B are matrix objects. Then `[A, B]` creates a new matrix object resulting from adjoining of A and B. Whereas `[A; B]` creates a new matrix object resulting from stacking of A and B. In fact A and B can be vector or scalar as long as they have appropriate dimensions for adjoining and stacking. For example

```
> A=[1,2;3,4]
```

	Col 1	Col 2
Row 1	1	2
Row 2	3	4

```
> [A,      [3; 7];
   A.sum(), A.sum().sum()]
```

	Col 1	Col 2	Col 3
Row 1	1	2	3
Row 2	3	4	7
Row 3	4	6	10

```
> [ones(3,1),identity(3,3)]
```

	Col 1	Col 2	Col 3	Col 4
Row 1	1	1	0	0
Row 2	1	0	1	0
Row 3	1	0	0	1

### 3.3.3 Selecting elements

Sometimes, you may want to select elements which satisfy certain condition. Here is an example:

```
> A = [1,2,3;4,5,6]

      Col 1      Col 2      Col 3
Row 1      1       2       3
Row 2      4       5       6
> A.select(A>2)

      i = 1      i = 2      i = 3      i = 4
      3         4         5         6
```

## 3.4 Computation

### 3.4.1 Friend function

There are several standard math functions available. The details about the ranges and error conditions can be found in your math library reference manuals. They are implemented as friend functions of class *Matrix*.

abs(A)	acos(A)	asin(A)	atan(A)	atan2(A,B)
ceil(A)	cos(A)	cosh(A)	erf(A)	erfc(A)
exp(A)	floor(A)	gammaln(A)	gamma(A,B)	int(A)
log(A)	log10(A)	max(A)	max(A,B)	min(A)
min(A,B)	mod(A,B)	pow(A)	round(A)	sin(A)
sinh(A)	sqrt(A)	tan(A)	tanh(A)	

The above friend functions are performed element-by-element, and the relevant objects A and B remain intact.

### 3.4.2 Member function

For a convenience, the standard math functions are also implemented as member functions.

A.abs()	A.acos()	A.asin()	A.atan()	A.atan2(B)
A.ceil()	A.cos()	A.cosh()	A.erf()	A.erfc()
A.exp()	A.floor()	A.gammaln()	A.gamma(B)	A.int()
A.log()	A.log10()	A.max()	A.max(B)	A.min()
A.min(B)	A.mod(B)	A.pow()	A.round()	A.sin()
A.sinh()	A.sqrt()	A.tan()	A.tanh()	

A.gammaln() returns the log of the gamma function value of A. The gamma function is defined as

$$\Gamma(x) = \int_0^{\infty} t^{(x-1)} e^{-t} dt$$

with  $\Gamma(1) = 1$ ,  $\Gamma(1/2) = \sqrt{\pi}$ , and  $\Gamma(x+1) = x\Gamma(x)$ .

A.gammaln() returns the incomplete gamma function value of A. The incomplete gamma function is defined as

$$\text{gammaln}(x, a) = \frac{\int_0^x t^{(a-1)} e^{-t} dt}{\Gamma(a)} \quad (a > 0)$$

### all

A.all() operates over the columns of A, returning a vector of 1's and 0's. If A is a one-row or one-column matrix, then the returned value would be a scalar. For example,

```
> A=[1,0,2;3,4,5]
```

	Col 1	Col 2	Col 3
Row 1	1	0	2
Row 2	3	4	5

```
> A.all()
```

i = 1	i = 2	i = 3
1	0	1

### any

`A.any()` operates over the columns of A, returning a vector of 1's and 0's. If A is a one-row or one-column matrix, then the returned value would be a scalar. For example,

```
> A=[1,0,2;3,4,5]
```

	Col 1	Col 2	Col 3
Row 1	1	0	2
Row 2	3	4	5

```
> A.any()
```

i = 1	i = 2	i = 3
1	1	1

### chol

`A.chol()` returns Cholesky decomposition for a real symmetric positive (semi)-definite matrix A, The object A remains intact. This Cholesky decomposition method takes only non-singular part with maximum rank of A into account, everything else are set to zeros. If A is positive definite, then `A.chol()` returns an lower triangular matrix L so that  $L*L' = A$ . For example,

```
> A=[5,3,2;3,3,0;2,0,2]
```

	Col 1	Col 2	Col 3
Row 1	5	3	2
Row 2	3	3	0
Row 3	2	0	2

```
> A.chol()
```

	Col 1	Col 2	Col 3
Row 1	2.23607	0	0
Row 2	1.34164	1.09545	0
Row 3	0.894427	-1.09545	0

### cond

`A.cond()` returns the condition number in norm2, which is the ratio of the largest singular value of A to the smallest. Object A remains intact.

### corrcoef

If each row of A and B is an observation and each column is a variable, `A.corrcoef(B)` returns a matrix whose the (i,j)'th element is the correlation coefficient between the i'th variable in A and the j'th variable in B. `A.corrcoef()` is identical to `A.corrcoef(A)`, returning a correlation coefficient matrix.

This member function is a user-defined function, it's loaded automatically into memory whenever you launch Matvec.

For example,

```
> A = [3,2,1;4,5,6;9,8,7];
> A.corrcoef()
```

	Col 1	Col 2	Col 3
Row 1	1	0.933257	0.741935
Row 2	0.933257	1	0.933257
Row 3	0.741935	0.933257	1

### covariance

.covariance(B) returns a matrix whose element (i,j) is the covariance between A(i,\*) and B(j,\*).

For example,

```
> A = [1,2; 3,4; 5,6];
> B = [6,5; 2,3;8,6];
> A.covariance(B)
```

	Col 1	Col 2
Row 1	2	1
Row 2	1	1

### det

A.det() returns the determinant of A based on LU decomposition. Object A remains intact.

### diag

A.diag(k), where A is a vector, returns a square matrix of order N+abs(k) with the elements of A on the k-th diagonal, where N is the size of vector A. k=0 (default) is the main diagonal, k > 0 is above the main diagonal and k < 0 is below the main diagonal.

A.diag(k), where A is a matrix, returns a vector formed from the elements of the k-th diagonal of A.

A.diag(B,C,D,...), where A,B,C,D,..., are all matrices, returns a block-diagonal matrix with A, B, C, ... on the diagonal

A.diag().diag() returns a diagonal matrix.

For example,

```
> c = [1,2;3,4]
```

	Col 1	Col 2
Row 1	1	2
Row 2	3	4

```
> c.diag()
```

i = 1	i = 2
1	4

```
> c.diag().diag()
```

	Col 1	Col 2
Row 1	1	0
Row 2	0	4

```
> c.diag([5])
```

	Col 1	Col 2	Col 3
Row 1	1	2	0
Row 2	3	4	0
Row 3	0	0	5

### Exponentials

A matrix exponential of a symmetric matrix A is

$$\exp(A) = \sum_{i=0}^{\infty} A^i / i! = P \exp(D) P'$$

where  $A = PDP'$  is the singular value decomposition of A. `A.mat_exp()` returns the matrix exponential of A. The matrix logarithm of a symmetric positive definite matrix is the inverse function of a matrix exponential and is given by

$$\ln(A) = P \ln(D) P'.$$

`A.mat_log()` returns the matrix logarithm of A. The partial derivatives of  $V = \exp(A)$  with respect to the elements of A,

$$\frac{\partial V}{\partial A} = \begin{pmatrix} \frac{\partial v_{11}}{\partial a_{11}} & \frac{\partial v_{11}}{\partial a_{12}} & \cdots & \frac{\partial v_{11}}{\partial a_{22}} & \cdots \\ \frac{\partial v_{12}}{\partial a_{11}} & \frac{\partial v_{12}}{\partial a_{12}} & \cdots & \frac{\partial v_{12}}{\partial a_{22}} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots \end{pmatrix},$$

is given by `V.mat_exp_der()`.

### eigen

`A.eigen()` returns a vector containing eigenvalues, and matrix A remain intact.

`A.eigen(U)` returns the same as `A.eigen()`, but with the eigenvectors brought out by matrix U so that  $A^*U = U^*D$  where D is a diagonal matrix with  $D(i,i)$ 's being the eigenvalues.

There are two basic algorithms: Jacobi and Householder, the latter is used in this package.

Note that if matrix A is non-symmetric, its eigenvalues/eigenvectors could be complex numbers, which are all ignored with certain warnings in Matvec.

For example,

```
> A = [3,2,1; 2,5,3; 1,3,4];
> A.eigen()
```

i = 1	i = 2	i = 3
2.38677	1.1944	8.41883

### flipr

`A.flipr()` return a matrix with the columns of A to be flipped from the left to the right.

This member function is a user-defined function, it's loaded automatically into memory whenever you launch Matvec.

For example,

```
> A = [1,2,3;4,5,6;7,8,9]
```

	Col 1	Col 2	Col 3
Row 1	1	2	3
Row 2	4	5	6
Row 3	7	8	9

```
> A.fliplr()
```

	Col 1	Col 2	Col 3
Row 1	3	2	1
Row 2	6	5	4
Row 3	9	8	7

### flipud

A.flipud() return a matrix with the rows of A to be flipped upside down.

This member function is a user-defined function, it's loaded automatically into memory whenever you launch Matvec.

For example,

```
> A = [1,2,3;4,5,6;7,8,9]
```

	Col 1	Col 2	Col 3
Row 1	1	2	3
Row 2	4	5	6
Row 3	7	8	9

```
> A.flipud()
```

	Col 1	Col 2	Col 3
Row 1	7	8	9
Row 2	4	5	6
Row 3	1	2	3

### ginv

A.ginv(mc) returns a matrix G, a generalized (or pseudo) inverse of A. The matrix G has the same dimensions as A' so that  $A*G*A = A$ ,  $G*A*G = G$  and AG and GA are Hermitian.

The argument mc is the method code, which is optional with default value 1. If mc = 0, then a method based on the singular value decomposition (SVD) is used. If mc = 1, then a method based on Cholesky decomposition is used. This method works only for symmetric positive semi-definite matrices, and is more efficient than the method based on SVD.

Any singular values less than a tolerance are treated as zero

For example,

```
> X=[1,1,0;
```

```
1,1,0;
```

```
1,1,0;
```

```
1,0,1;
```

```
1,0,1];
```

```
> A = X'*X
```

	Col 1	Col 2	Col 3
Row 1	5	3	2
Row 2	3	3	0
Row 3	2	0	2

```
> A.ginv()
```

	Col 1	Col 2	Col 3
Row 1	0.0925926	0.0185185	0.0740741
Row 2	0.0185185	0.203704	-0.185185
Row 3	0.0740741	-0.185185	0.259259

```
> A.ginv(1)
```

	Col 1	Col 2	Col 3
Row 1	0.5	-0.5	0
Row 2	-0.5	0.833333	0
Row 3	0	0	0

### identity

- `identity(m,n)` creates an identity matrix of order m by n.
- `A.identity()` resets matrix A to be an identity.
- `A.identity(n)` resets matrix A to be an identity of order n by n.
- `A.identity(m,n)` resets matrix A to be an identity of order m by n.

### input

`A.input("<filename>",m,n)` inputs an m by n matrix whose elements are input from "<filename>" row-by-row. Each row can occupy several lines, however, each row must start from a new line. Then columns in "<filename>" must be separated with blanks or tabs.

### inv

`A.inv()` returns the inverse of A based on LU decomposition. Matrix A must be non-singular, otherwise run-time error will result in. For example,

```
> A = [3,2,1; 2,5,3; 1,3,4]
```

	Col 1	Col 2	Col 3
Row 1	3	2	1
Row 2	2	5	3
Row 3	1	3	4

```
> A.inv()
```

	Col 1	Col 2	Col 3
Row 1	0.458333	-0.208333	0.0416667
Row 2	-0.208333	0.458333	-0.291667
Row 3	0.0416667	-0.291667	0.458333

### kron

`A.kron(B)`, which is the same as `A@B`, is the Kronecker tensor product of matrices A and B. The result is a large matrix formed by taking all possible products between the elements of A and those of B. For example, if A is 2 by 3, then `A.kron(B)` is

```
[ A(1,1)*B  A(1,2)*B  A(1,3)*B
  A(2,1)*B  A(2,2)*B  A(2,3)*B ]
```

### logdet

`A.logdet()` returns the natural log of the g-determinant for a real symmetric positive (semi)definite (psd) matrix. A remains intact. "g" in g-determinant means it takes non-singular part of matrix A with maximum rank into account.



**max**

`A.max()` returns a vector whose each element is the largest element of each column of `A`. If `A` is a one-row or one-column matrix, then the returned value would be a scalar.

If both `A` and `B` are matrices (one of them could be a scalar), then `A.max(B)` returns a matrix of the same size as `A` or `B` with resulting element(`i,j`) being the largest element among `A(i,j)` and `B(i,j)`.

For example,

```
> A=sample(2,2)
```

	Col 1	Col 2
Row 1	0.0100414	0.578454
Row 2	0.973198	0.792847

```
> B=sample(2,2)
```

	Col 1	Col 2
Row 1	0.619559	0.927987
Row 2	0.89682	0.990276

```
> A.max()
```

	i = 1	i = 2
	0.973198	0.792847

```
> A.max(B)
```

	Col 1	Col 2
Row 1	0.619559	0.927987
Row 2	0.973198	0.990276

**mean**

`A.mean()` returns a vector containing average values for each column of `A`. If `A` is a one-row or one-column matrix, then the returned value would be a scalar.

**min**

`A.min()` returns a vector whose each element is the smallest element of each column of `A`. If `A` is a one-row or one-column matrix, then the returned value would be a scalar.

If both `A` and `B` are matrices (one of them could be a scalar), then `A.min(B)` returns a matrix of the same size as `A` or `B` with resulting element(`i,j`) being the smallest element among `A(i,j)` and `B(i,j)`.

For example,

```
> A=sample(2,2)
```

	Col 1	Col 2
Row 1	0.0100414	0.578454
Row 2	0.973198	0.792847

```
> B=sample(2,2)
```

	Col 1	Col 2
Row 1	0.619559	0.927987
Row 2	0.89682	0.990276

```
> A.min()
```

	i = 1	i = 2
	0.0100414	0.578454

```
> A.min().min()
0.0100414
> A.min(B)
```

	Col 1	Col 2
Row 1	0.0100414	0.578454
Row 2	0.89682	0.792847

**mod**

`A.mod(B)` returns a matrix with elements being the mod of corresponding elements in A and B. Both A and B must be the same size except B can be a scalar.

**norm**

For matrices:

- `A.norm()` is the largest singular value of A, namely `A.svd().max()`.
- `A.norm(2)` is the same as `A.norm()`.
- `A.norm(1)` is the 1-norm of A, the largest column sum, namely `A.abs().sum().max()`.
- `A.norm("inf")` is the infinity norm of A, the largest row sum, namely `A.t().abs().sum().max()`.
- `A.norm("fro")` is the F-norm, defined as `(A'*A).diag().sum().sqrt()`.

For vectors:

- `A.norm(p) = (A.abs().^p).sum().^(1/p)`
- `A.norm() = A.norm(2)`.
- `A.norm("inf") = A.abs().max()`.
- `A.norm("-inf") = A.abs().min()`.

If A is a scalar, then it will be treated as a single element vector.

**ones**

`A.ones(m,n)` first resize and then set each element of matrix A to be unit  
`A.ones()` resets each element of matrix A to be unit.

**pow**

`A.pow(B)` returns a matrix with elements being the elements in A with the power of the corresponding elements in B. Both A and B must be the same size except B can be a scalar.

**product**

`A.product()` returns a vector containing product values for each column of A. If A is a one-row or one-column matrix, then the returned value would be a scalar.

**rank**

`A.rank()` returns rank of an object A based on singular value decomposition.

**reshape**

`A.reshape(m,n)` returns a new matrix of `m` rows and `n` columns, whose elements are taken from the matrix `A` row-by-row. If the matrix `A` doesn't have enough elements, then the operation cycles back to the beginning of `A` to get enough values for the new matrix. For example,

```
> A = [1,0,0,0];
> B = A.reshape(3,3)
```

	Col 1	Col 2	Col 3
Row 1	1	0	0
Row 2	0	1	0
Row 3	0	0	1

```
> C = A.reshape(3,4)
```

	Col 1	Col 2	Col 3	Col 4
Row 1	1	0	0	0
Row 2	1	0	0	0
Row 3	1	0	0	0

**resize**

`A.resize(m, n)`, Matrix `A` is resized to be `m`-by-`n`. The contents in `A` could be garbage.

**rot90**

`A.rot90(k)` rotate the matrix `A` counterclockwise `k*90` degrees.

This member function is a user-defined function, it's loaded automatically into memory whenever you launch Matvec.

For example,

```
> A = [1,2,3;4,5,6;7,8,9]
```

	Col 1	Col 2	Col 3
Row 1	1	2	3
Row 2	4	5	6
Row 3	7	8	9

```
> A.rot90()
```

	Col 1	Col 2	Col 3
Row 1	3	6	9
Row 2	2	5	8
Row 3	1	4	7

```
> A.rot90(1)
```

	Col 1	Col 2	Col 3
Row 1	3	6	9
Row 2	2	5	8
Row 3	1	4	7

**sample**

`A.sample()` replaces its elements with random numbers from `UniformDist(0.0,1.0)`. `A.sample(m,n)` first resizes matrix `A` to `m` by `n`, then replaces its elements with random numbers from `UniformDist(0.0,1.0)`.

**save**

`A.save("<filename>", "<option>")` saves an ASCII copy of the contents of the object `A`. The second argument "`<option>`" is optional with default value "`noreplace`"; it specifies the output mode with the following possible values: "`out`", "`app`", and "`noreplace`". The mode "`out`" means out by force and overwrite the existing file if it exists. The mode "`app`" causes all output to that file to be appended to the end. The mode "`noreplace`" causes failure if the file does already exist.

**select**

`A.select(B)` returns a vector whose element are selected from `A(i,j)` based on the corresponding element `B(i,j)` is nonzero or zero.

For example,

```
> A = sample(3,3)
# I want all elements A(i,j) that are >= 0.5
> A.select(A >= 0.5)
```

i = 1	i = 2	i = 3	i = 4
0.865423	0.912348	0.967485	0.641004

**splines**

`T.splines(k,typei)` returns a  $n \times nk$  cubic spline design where `T` is a  $n \times 1$  matrix of design points, `k` is a  $nk \times 1$  matrix of knots. The optional type parameter determines the basis type used:

- `type=0` uses the parameterization  $g(k_i) = g_i$  and is the default parameterization.
- `type=1` uses the parameterization  $g(k_i) = \beta_o + \beta_1 k_i + Z'_i u$  where  $\beta_o + \beta_1 t$  is a straight line component and  $u$  is a  $nk - 2$  "random" deviation vector whose magnitude determines the smoothness of the fitted spline.

```
knots=[0 .5 1.25 2 2.75 3.5 4 ].t();
g=20/(1+exp(-2*(knots-2.5)))
T=[0:40].t()/10;
X=T.splines(knots)
y=X*g;
[T y]
P=Plotter();
P.plot([T y])
sleep()
```

**solve**

`A.solve(b, stopval, relax, mxiter)` returns the solution (vector or matrix) of the linear equations  $Ax = b$ , where `b` could be a vector or matrix object.

The first argument, which is the right hand side of linear equations, is mandatory. The rest of arguments are optional. The argument `stopval` is the criterion value to stop the iteration with its default 0.001. When the argument `relax = 1.0` (default), the linear equations is solved using Gauss-Seidel iteration; otherwise using the method of successive overrelaxation (SOR). The last argument `mxiter` is the maximum number of iterations allowed with its default 100,000.

Note that `A.solve(b)` is much more efficient than `A.inv()*b`.

**sort**

If A is a matrix, A.sort() sorts each column of A in an ascending order.

If A is a vector, A.sort() sorts the vector in an ascending order.

If you want to sort it in descending order, the following will do the job:

```
> a = Vector(3,2,4);
> b = a.sort()

      i = 1      i = 2      i = 3
        2        3        4

> b(3:1:-1)      # sort it in descending order
      i = 1      i = 2      i = 3
        4        3        2
```

**std**

A.std() returns a vector containing the standard deviations for each column of A.

This member function is a user-defined function, it's loaded automatically into memory whenever you launch Matvec.

For example,

```
> A = [3,2,1;4,5,6;9,8,7];
> A.std()

      i = 1      i = 2      i = 3
    3.21455      3      3.21455
```

**sum**

A.sum() returns a vector containing sum values for each column of A. If A is a one-row or one-column matrix, then the returned value would be a scalar.

**sumsq**

A.sumsq() returns a vector containing sum square values for each column of A. If A is a one-row or one-column matrix, then the returned value would be a scalar.

**svd**

The singular value decomposition of an m-by-n matrix A is a decomposition

$$A = U*W*V'$$

where U is m-by-n with  $U'*U = I$ , W is an n-by-n diagonal matrix and V is an n-by-n orthogonal matrix.

Both A.svd() and A.svd(U,V) returns a vector containing diagonals in W, but the latter has U and V which must be declared as matrix objects before calling.

If A is a scalar, then it will be treated as a 1-by-1 matrix.

For example,

```
> a=[1,2,3;2,9,4;2,1,8];
> u=[]; v=[];
> a.svd(u,v)

      i = 1      i = 2      i = 3
    12.0953    0.0538603    6.14008
```

```

> u
u =

      Col 1      Col 2      Col 3
Row 1  -0.304833  -0.946777   0.103397
Row 2  -0.770928   0.181542  -0.610502
Row 3  -0.559237   0.265813   0.785237

> v
v =

      Col 1      Col 2      Col 3
Row 1  -0.24515   -0.966676   0.073756
Row 2  -0.670281   0.114035  -0.733293
Row 3  -0.700446   0.229204   0.6759

```

**sweep**

A.sweep(i1,i2) sweeps matrix A on the pivots indicated by i1 through i2 to produce a new matrix.

For example, suppose that A is partitioned into

$$\begin{bmatrix} R & S \\ T & U \end{bmatrix}$$

such that R is q by q and non-singular, U is m-q by n-q. then A.sweep(1,q) returns

$$\begin{bmatrix} R^{-1} & R^{-1}S \\ -TR^{-1} & U - TR^{-1}S \end{bmatrix}$$

It is obvious that A.sweep() can be used for inverting matrix. It seems to me that this is best one to inverse matrix in the sense of memory requirement for non-singular full storage matrix.

**t**

A.t() returns the transpose of A, which remains intact. Another way to transpose a matrix is to use Matvec operator '. Thus A.t() is equivalent to A'.

**trace**

A.trace(), trace of matrix A, A isn't destroyed. A is not necessary be to square.

**tril**

A.tril(k) returns the elements on and below the k-th diagonal of A where k is optional. k=0 (default) is the main diagonal, k > 0 is above the main diagonal and k < 0 is below the main diagonal.

This member function is a user-defined function, it's loaded automatically into memory whenever you launch Matvec.

For example,

```

> A = [1,2,3;4,5,6;7,8,9]

      Col 1      Col 2      Col 3
Row 1      1      2      3
Row 2      4      5      6
Row 3      7      8      9

> A.tril()

```

	Col 1	Col 2	Col 3
Row 1	1	0	0
Row 2	4	5	0
Row 3	7	8	9

```
> A.tril(-1)
```

	Col 1	Col 2	Col 3
Row 1	0	0	0
Row 2	4	0	0
Row 3	7	8	0

**triu**

A.triu(k) returns the elements on and above the k-th diagonal of A where k is optional. k=0 (default) is the main diagonal, k > 0 is above the main diagonal and k < 0 is below the main diagonal.

This member function is a user-defined function, it's loaded automatically into memory whenever you launch Matvec.

For example,

```
> A = [1,2,3;4,5,6;7,8,9]
```

	Col 1	Col 2	Col 3
Row 1	1	2	3
Row 2	4	5	6
Row 3	7	8	9

```
> A.triu()
```

	Col 1	Col 2	Col 3
Row 1	1	2	3
Row 2	0	5	6
Row 3	0	0	9

```
> A.triu(1)
```

	Col 1	Col 2	Col 3
Row 1	0	2	3
Row 2	0	0	6
Row 3	0	0	0

**variance**

A.variance() returns a vector containing variance values for each column of A.

**vec**

A.vec(), where A is a matrix, returns a Vector converting from matrix A column-by-column. For example,

```
> A=[1,2;3,4]
```

	Col 1	Col 2
Row 1	1	2
Row 2	3	4

```
> A.vec()
```

i = 1	i = 2	i = 3	i = 4
1	3	2	4

**vech**

`A.vech()`, where `A` is a matrix, returns a vector converting from lower triangular part of matrix `A` column-by-column. For example,

```
> A=[1,2;3,4]
```

	Col 1	Col 2
Row 1	1	2
Row 2	3	4

```
> A.vech()
```

i = 1	i = 2	i = 3
1	3	4

**zeros**

`zeros(m,n)` creates a matrix object of `m` by `n`, with each element being zero.

`A.zeros()` resets each element of matrix `A` to be zero.

## 3.5 Special Matrices

In `Matvec`, there are 11 special matrices: `ones`, `zeros`, `identity`, `sample`, `hilb`, `invhilb`, `hankel`, `vander`, `hadamard`, `pascal`, `toeplitz`. The first four are builtin functions, while the rest are defined in macro package “`special_matrix`”.

### 3.5.1 ones

`ones(m,n)` creates a matrix object of order `m` by `n`, with each element being unit. For example,

```
> A = ones(2,3)
```

	Col 1	Col 2	Col 3
Row 1	1	1	1
Row 2	1	1	1

### 3.5.2 zeros

`zeros(m,n)` creates a matrix object of order `m` by `n`, with each element being zero. For example,

```
> A = zeros(2,3)
```

	Col 1	Col 2	Col 3
Row 1	0	0	0
Row 2	0	0	0

### 3.5.3 identity

`identity(m,n)` creates an identity matrix of order `m` by `n`. For example,

```
> A = identity(3,3)
```

	Col 1	Col 2	Col 3
Row 1	1	0	0
Row 2	0	1	0



```

      Row 3          0          0          1
> A = identity(2,3)

      Col 1      Col 2      Col 3
Row 1          1          0          0
Row 2          0          1          0

```

### 3.5.4 sample

`sample(m,n)` returns a random sample from `UniformDist(0.0,1.0)`, stored in a matrix of m by n. For example,

```

> A = sample(2,3)

      Col 1      Col 2      Col 3
Row 1  0.792847  0.619559  0.927987
Row 2  0.89682   0.990276  0.266462

```

### 3.5.5 other special matrices

Other special matrices in `Matvec` are `hilb`, `invhilb`, `hankel`, `vander`, `hadamard`, `pascal`, `toeplitz`. They are provided in a macro package `special_matrix`. See details in Chapter 9, section 9.3.



## Chapter 4

# Program Flow Control

The program flow-control statements provided by Matvec are quite rich: *if*, *while*, *for*, *break*, *continue*, *repeat*, and *null*. They are similar to those in the C/C++ language:

A BLOCK construction is defined as either a single statement or a sequence of statements (not necessary to be in the same line) enclosed with braces. For instance

```
x = 8;
```

is a BLOCK, and

```
{  
    a = 6;  
    b = sample(3,5);  
}
```

is a BLOCK, too.

### 4.1 If–Endif and If–Else Statements

The *if* statement has two forms:

```
if (expression) BLOCK1 endif  
if (expression) BLOCK1 else BLOCK2
```

The expression must evaluate to a scalar, otherwise a run-time error will result.

In the first form of *if* statement, it simply performs a test on the expression in the parenthesis, and if the expression evaluates to a non-zero scalar, the statement(s) in BLOCK will be executed. The *endif* is mandatory to tell Matvec interpreter not to expect an else-branch.

In the second form of *if* statement, it performs a test on the expression in the parenthesis, and if the expression is true then executes BLOCK1 otherwise execute BLOCK2.

A nested *if* is an *if* that is the part of a BLOCK belonging to another *if* or *else*. In Matvec an *else*-branch always refers to the nearest *if* statement that is within the same block as the *else* and is not already associated with an *if*. For example,

```
if (i) {  
    if (j) BLOCK1 endif;  
    if (k) BLOCK2 else BLOCK3  
}  
else {  
    BLOCK4  
}
```

Note that Matvec does not have *elseif* statement, nor does C/C++. Matvec, however, allow users to form an *if-else-if* ladder:

```
if (expr)
    BLOCK1
else if (expr)
    BLOCK2
else if (expr)
    BLOCK3
.
.
.
else
    BLOCKn
```

The final statement in this *if-else-if* ladder must be either *else* or *endif*. The expr-conditions are evaluated from top downward. As soon as a true condition is found, the BLOCK associated with it is immediately executed and the rest of the ladder is bypassed. If none of the expr-conditions are true, then BLOCKn is executed if this ladder has a final *else* branch, otherwise no action takes place if this ladder ends with a *endif*. For example,

```
if (A.class() == "Scalar") {
    if (A < 0) {
        retval = -1;
    }
    else if (A == 0) {
        retval = 0;
    }
    else {
        retval = 1;
    }
}
endif;
```

The any and all functions may be useful with *if* statement as shown below:

```
if (A.any().any()) {
    "there are non-zero elements in matrix A"
}
else {
    " all elements in matrix A are zero"
}
```

## 4.2 For Statement

The *for* statement has also two forms:

```
for (initialization; condition; increment) BLOCK
for (i in expression) BLOCK
```

The first form of *for* statement is similar to that in C/C++. In general, **initialization** is generally an assignment statement that is used to set the loop control variable. The **condition** is usually a relational expression that determines when the loop exits. The **increment** defines how the loop control variable changes each time the loop is repeated. You must separate these three sections by semicolons. The *for* loop continues to execute until the condition becomes false. The execution resumes on the statement following *for*. Note

that all those three sections can be omitted simultaneously, thus `for (; ;)` BLOCK construction provides user an infinite loop. The *break* statement is the only way to exit such an infinite loop.

The second form of *for* statement is similar to that in Maple language. The expression must evaluate to a either matrix, or vector, or list, otherwise a run-time error will result. The *for* statement simply executes the BLOCK the N times where N is the size of expression unless it encounters the *break* statement, which will terminate the nearest for-loop execution. The *for* statement can be nested, too. For instance,

```
> A = zeros(3,4);
> for (i=0; i<3; i++) {
    for(j in 1:4) A(i,j) = i+j;
}
> A
```

	Col 1	Col 2	Col 3	Col 4
Row 1	2.00000	3.00000	4.00000	5.00000
Row 2	3.00000	4.00000	5.00000	6.00000
Row 3	4.00000	5.00000	6.00000	7.00000

## 4.3 While Statement

The while statement is very similar to that in C/C++.

```
while (expression) BLOCK
```

If the expression evaluates to anything rather than zero, then the BLOCK is executed, otherwise goes to the statement right after the BLOCK.

For example,

```
> i=1; j=0;
> while(i > 10) {
    j += i++;
}
> j
45
```

## 4.4 Repeat–Until Statement

The repeat statement is very similar to that in PASCAL language.

```
repeat BLOCK until (expression)
```

the BLOCK is executed until the expression evaluates to anything rather than zero. Then, it goes to the statement right after the expression.

For example,

```
> i=1; j=0;
> repeat {
    j += i++;
} until (i >= 10)
> j
45
```



## Chapter 5

# File Stream Control

Matvec provides a class of input/out stream, which is a substantially limited C++ I/O stream system. A stream is a logical object associated with a physical device such as disk.

When output is performed, data is not immediately written to the physical device associated with the stream. Instead, data is stored in an internal buffer until the buffer is full. However, data in the buffer can be forced out to disk before the buffer is full by using either `endl()` or `flush()`. Note that buffering is used to improve performance. Flushing each line of output using either `endl()` or `flush()` decreases efficiency.

### 5.1 Standard File Streams

There are three standard file streams created automatically each time when Matvec is invoked: `cout`, `cin`, `cerr`.

#### 5.1.1 `cout`

not yet available

#### 5.1.2 `cin`

not yet available

#### 5.1.3 `cerr`

not yet available

### 5.2 User Specified File Stream

#### 5.2.1 Creation

`f = FileStream("<filename>", "<mode>")` creates an object of class `FileStream` and assign to `f`. The "<filename>" can be either relative or absolute, and the "<mode>" must be one the following or appropriate combination: `out`, `in`, `app`, `noreplace`.

#### 5.2.2 Example

```
> f = FileStream("try.out", "out");
> f.out("I am testing MATVEC file stream\n");
> Pi=3.14159;
> f.out("Pi = ", Pi, " 2*Pi = ", 2*Pi, "\n");
```

```
> f.close()
> $more try.out
I am testing MATVEC file stream
Pi = 3.14159 2*Pi = 6.28319
```

## 5.3 Member Function

### 5.3.1 open

`f.open("<filename>", "<mode>")` opens the file stream `f`. The "`<filename>`" can be either relative or absolute, and the "`<mode>`" must be one the following or appropriate combination: `out`, `in`, `app`, `noreplace`.

### 5.3.2 reopen

`f.reopen()` reopens the file stream for either writing or reading. not yet available

### 5.3.3 close

`f.close()` closes the file stream.

### 5.3.4 set

`f.set("<flag>", n)` sets value `n` for the associated flag: "`width`", "`precision`", "`format`", "`fill`".

### 5.3.5 out

`f.out(arg1, arg2, ...)` writes out `arg1`, `arg2`, ... to the file stream.

### 5.3.6 in

`f.in(arg1, arg2, ...)` reads in `arg1`, `arg2`, ... from the file stream.  
not yet available

### 5.3.7 getline

`f.getline()` reads and returns a string line from the file stream.  
not yet available

### 5.3.8 eof

`f.eof()` returns nonzero when the end of the file stream has been reached; otherwise it returns zero.

### 5.3.9 endl

`f.endl()` immediately flushes data in the buffer to its destination: either disk or screen.

### 5.3.10 flush

`f.flush()` forces the data in the buffer to be physically written to its destination (either disk or screen) in the same way as `endl()`.

### 5.3.11 rewind

`f.rewind()` resets the position indicator to the beginning of the file stream. Note that `f` must be a read/write `FileStream`.



## Chapter 6

# Time Control

Though I do not know what time and date exactly mean, I did build several conventional functions related to time and date.

### 6.1 Digital Clock

#### 6.1.1 Time

`time()` returns the current system calendar time, which is a double precision number. It is a builtin function. For example,

```
> this.parameter("OUTPUT_PRECISION",10);
> time()
      815691299
```

#### 6.1.2 Localtime

`localtime()` returns a vector of local time with seven components: year, month, day-of-month, day-of-year, hour, minute, and second. Such a vector will be referred to as time vector (tv). Whereas `localtime(t)` converts the calendar time represented by `t` into the local time vector. It is a builtin function. For example,

```
> t = time()
      8.15692e+08

> localtime()

      i = 1      i = 2      i = 3      i = 4      i = 5      i = 6
      1995       11        6        3101        4         9

      i = 7
      13

> localtime(t)

      i = 1      i = 2      i = 3      i = 4      i = 5      i = 6
      1995       11        6        310        14        9

      i = 7
      5
```

### 6.1.3 Gmtime

`gmtime()` is the same as `localtime()` except for Greenwich mean time. So is `gmtime(t)`.

### 6.1.4 Asctime

`asctime(tv)` converts the time vector (`tv`) to a character string. It is a builtin function. For example,

```
> asctime(localtime())
    Sun Nov  6 14:16:34 1995
```

### 6.1.5 Dftime

`dftime(t1,t0)` returns the difference in second between two times `t1` and `t0`, where `t1` and `t0` must be values returned by `time()`. It is a builtin function. The following script times a loop using `time()` and `dftime()`:

```
> # timing a loop using time() and dftime()
> t0 = time();
> for (i=0; i<1000; i++) sample(100,100);
> t1 = time();
> dftime(t1,t0)
    41
```

### 6.1.6 Mkttime

`mktime(tv)` converts the time vector, `tv`, to system calendar time. It is a builtin function. The following statement returns the system calendar time of the current local time vector:

```
> mktime(localtime())
    8.15697e+08
```

### 6.1.7 Date

`date()` is a macro function, equivalent to `asctime(localtime())`. For example

```
> date()
    Sun Nov  6 14:24:50 1995
```

### 6.1.8 Ctime

`ctime(t)` converts the calendar time represented by `t` to a string. It is a macro function, equivalent to `asctime(localtime(t))`. For example,

```
> t = time();
> ctime(t)
    Sun Nov  6 14:30:13 1995
```

### 6.1.9 Clock

`clock()` is a macro friend function, equivalent to `localtime()`.

## 6.2 Western Calendar

### 6.2.1 Leap year

`leapyear(year)` is a macro friend function. It returns 1 if the argument `year` is a leap year, otherwise return 0. For example,

```
> # Is 1990 a leap year?  
> leapyear(1990)  
0  
> # Is 1996 a leap year?  
> leapyear(1996)  
1
```



## Chapter 7

# User-Defined Function

### 7.1 Syntax

```
function obj.foo(args) { <MATVEC statements> }  
function foo(args) { <MATVEC statements> }  
protected function obj.foo(args) { <MATVEC statements> }  
protected function foo(args) { <MATVEC statements> }
```

### 7.2 Description

User-defined functions in Matvec can be constructed or defined) by the above four forms. The first form is so-called a user-defined member function with respect to object *obj*, whereas the second form is so-called a user-defined friend function. The third and fourth are the same as the first two except that functions are protected. In another word function definition in the last two forms can not be overwritten.

Usually, the definition of a user-defined functions are stored in a disk file. Before calling them, we have to load them into the computer memory using the Matvec builtin function `input`:

```
input("<filename>")
```

User-defined functions in Matvec are treated as objects, in a manner similar to other objects such as Matrix, Vector. Thus, in principle, users can assign/copy them like ordinary variables, but users should avoid to use this feature.

By default all variables within a function are global except those symbolic arguments in function header and those declared in local statements: `local(A,B,C,...)`;

Note that *local* statement can appear anywhere only within function definition, and multiple *local* statements are also allowed.

Function execution will terminated right after return statement. the returning value of a function is the last expression evaluated in the function execution. The return statement in the function is optional.

### 7.3 Scope

### 7.4 Recursion

not working yet so far.

### 7.5 Examples

```
protected function A.sign()
```

```

{
  #
  # Return 1,  if element >= 0
  # Return 0,  if element == 0
  # Return -1, if element < 0
  #
  # Tianlin Wang at UIUC, Fri Dec 30 16:10:38 CST 1994
  #

  local(i, j, retval);
  if (A.class() == "Scalar") {
    if (A > 0)  retval = 1;  endif;
    if (A == 0) retval = 0;  endif;
    if (A < 0)  retval = -1; endif;
  }
  else if (A.class() == "Vector") {
    retval = Vector();
    retval.resize(A.size());
    for (i in 1:A.size()) {
      if (A(i) > 0) retval(i) = 1;  endif;
      if (A(i) == 0) retval(i) = 0;  endif;
      if (A(i) < 0) retval(i) = -1; endif;
    }
  }
  else if (A.class() == "Matrix") {
    retval = Matrix(A.nrow(),A.ncol());
    for (i in 1:A.nrow()) {
      for (j in 1:A.ncol()) {
        if (A(i,j) > 0) retval(i,j) = 1;  endif;
        if (A(i,j) == 0) retval(i,j) = 0;  endif;
        if (A(i,j) < 0) retval(i,j) = -1; endif;
      }
    }
  }
  else {
    retval = 0;
    error("sign only works for numeric objects");
  }
  return retval;
}

```

## Chapter 8

# 2D/3D Object-Oriented Plotting

The 2-dimension and 3-dimension plotting within Matvec interface are currently using external shell program: *gnuplot*. This means *gnuplot* must be installed in the same machine where Matvec resides.

### 8.1 Creation

`p = Plotter()` creates an object of class `Plotter`.

### 8.2 Member Function

#### 8.2.1 plot

`p.plot("literal function such as sin(x)")` plots function  $y=\sin(x)$  on your X11 terminal.

`p.plot(x,y,"<options>")` plots vectors  $x$  versus  $y$  on your X11 terminal. "options" is optional, with possible value: {title '<title>'}{with <style>} where <style> can be one of the following six styles: lines, point, linespoints, impulses, dots, step, or errorbars.

#### 8.2.2 plot3D

`p.plot3D("literal function such as sin(x)+cos(y)")` plots function  $z = \sin(x) + \cos(y)$  on your X11 terminal.

`p.plot3D(x,y,z,"<options>")` plots vectors  $x$ ,  $y$  versus  $z$  on your X11 terminal "<options>" is optional, with possible value: {title '<title>'}{with <style>} where <style> can be one of the following six styles: lines, point, linespoints, impulses, dots, step, or errorbars.

#### 8.2.3 replot

`p.replot()` replot the previous data or function after setting title, labels, etc.

#### 8.2.4 set

`p.set("key value")` where key can be one of followings:

```

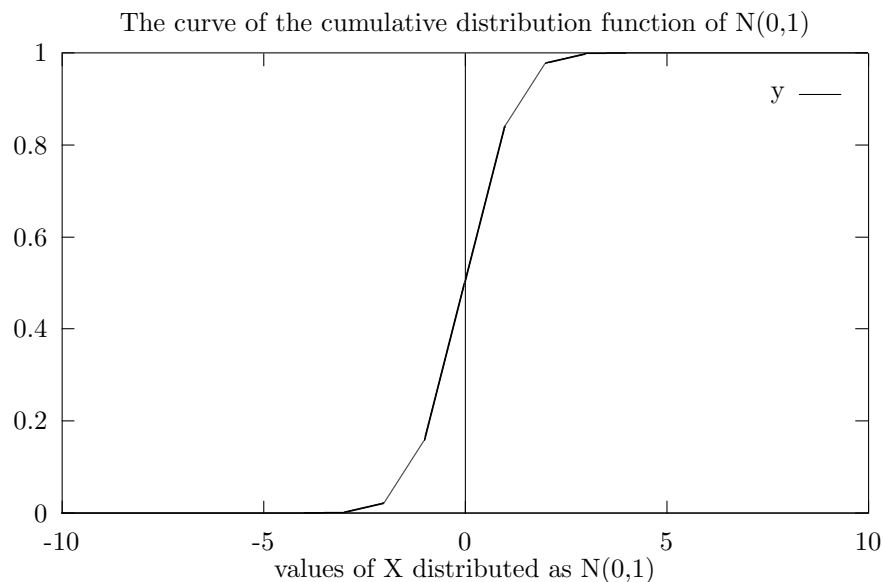
angles          {degree | radians}
arrow           <tag> {from sx,sy,sz} {to ex,ey,ez}
autoscale       <axes>
border
boxwidth        <width>
clabel
clip            <clip_type>
contour         {base | surface | both}
data sytle      <style_choice>
dummy           <dummy_var> {,<dummy_var>}
format          {<axes>} {'<format_string>'}
function style  <style_choice>
hidden3d
key             <x>,<y>{,<z>}
label           {<tag>}{'<label_text>'}{at x,y{,z}}{<justification>}
logscale        <axes><base>
mapping         {cartesian | spherical | cylindrical}
offsets         <left>,<right>,<top>,<bottom>
output          {'<filename>'}
parametric
polar
rrange          [{<rmin> : <rmax>}]
samples         <samples_1> {,<samples_2>}
size            {<xscale>,<yscale>}
surface
terminal        {<terminal_type>}
tics            {<direction>}
time            {<xoff>} {,<yoff>}
title           {'<title_text>'} {<xoff>} {,<yoff>}
trange          [{<tmin> : <tmax>}]
urange          [{<umin> : <umax>}]
view            <rot_x> {,<rot_z>} {,<scale>} {,<scale_z>}}
vrangle         [{<vmin> : <vmax>}]
xlabel          {'<label>'} {<xoff>} {,<yoff>}
xrange          [{<xmin> : <xmax>}]
xtics           [{<start>,<incr> {,<end>}}|{...}]
xmtics
xzeroaxis
ylabel          {'<label>'}{<xoff>}{,<yoff>}
yrange          [{<ymin> : <ymax>}]
yticks
ymticks
yzeroaxis
zero            <expression>
zlabel          {'<label>'} {<xoff>} {,<yoff>}
zrange          [{<zmin> : <zmax>}]
zticks
zmticks

```

### 8.2.5 save

`p.save("<fname>","<format>")` save the graph into your disk in the specified format. The second argument is optional with default value "postscript". The common graphic formats are postscript, latex, fig, eepic,



Figure 8.1: The curve of the cumulative distribution function of  $N(0,1)$ 

emtex, mf, pbm, x11, etc.

### 8.2.6 open

`p.open()` open (re-open) a shell for external plotting program

### 8.2.7 close

`p.close()` close the graph window on X11 terminal. Note that after closing the graph window, all settings such as title, labels are also cleared.

## 8.3 Examples

### 8.3.1 Example 1

Try the following example

```
> N = StatDist("Normal",0,1);
> x = -10:10;
> y = N.cdf(x);

> p = Plotter();
> p.set("xlabel 'values of X distributed as N(0,1)'" )
> p.plot(x,y);
```

The picture from the above Matvec script is shown in Fig. 8.1:

### 8.3.2 Example 2

```
> p = Plotter();
> p.plot3D("sin(x)+cos(y)");
> p.save("myfig.ps");
```

### 8.3.3 Example 3

The following Matvec script draw the graph of normal pdf function for two sets of parameters.

```
D1 = StatDist("Normal",0,1);
D2 = StatDist("Normal",2,0.5);

x = -4:4:0.1;
xy1 = [x; D1.pdf(x)]';
xy2 = [x; D2.pdf(x)]';
xy1.save("Normal.dat1",1);
xy2.save("Normal.dat2",1);

curve1 = " 'Normal.dat1' title 'N(0,1)' with lines 1,";
curve2 = " 'Normal.dat2' title 'N(2,0.5)' with lines 4";
curve = curve1 + curve2;

P = Plotter();
P.set("nozeroaxis");
P.set("size 4/5.0, 2.5/3.0");
P.set("title 'Normal Probability Density Function (pdf)' ");
P.set("xlabel 'x'");
P.set("ylabel 'pdf(x)'");
P.set("key -2.0,0.5");

P.plot(curve);
```

The graph is shown on page 90.

### 8.3.4 Example 4

The following Matvec script draw the graph of the pdf of  $b(25, 0.15)$ .

```
D = StatDist("Binomial",25,0.15);

x = -0.5:19.5;
xy = [x; D.pdf([0:20])]';

xy.save("Binomial.dat",1);

P = Plotter();
P.set("title 'Binomial Probability Density Function (pdf)' ");
P.set("nozeroaxis");
P.set("format y '%3.2f' ");
P.set("xtics 0, 1.0, 13");
P.set("xrange [-0.5 : 12.5]");
P.set("size 4/5.0, 2.5/3.0");
P.set("xlabel 'k'");
P.set("ylabel 'pdf(k)'");

P.plot(" 'Binomial.dat' title 'b(25,0.15)' with steps");
```

The graph is shown on page 103.

## Chapter 9

# Macro Packages

A macro package is a single plain text file, consisting of a set of user-defined functions to accomplish a specific task. Before accessing those user-defined functions in a macro package, you first have to load the package into your computer memory with Matvec function

```
package("<package_name>")
```

Each macro package should have an appropriate package name (related to the task) with the extension `.mv`. A package\_name is referred to as an absolute name if it contains either at least one slash (/) or file extension `.mv`, otherwise it is referred to as a relative name. If a package\_name is a relative name, Matvec will only search for it in the directory `$(MATVECHOME)/packages/contrib`. If a package name is an absolute name, it must contain the full information including pathname and file extension. If you want to load a macro package called `mypackage` in the current working directory, the following trick works:

```
package("./mypackage")
```

However, users are strongly encouraged to name your macro package with file extension `.mv`.

A variety of macro packages are provided with the standard Matvec distribution. They usually reside in the `MATVECHOME/packages/contrib` directory. These macro packages are called the contribution packages because most of them are contributed by Matvec experienced users. They are the good resources for Matvec beginners to learn how to properly and efficiently use Matvec system. So take a close look at.

Any good macro packages written by users can be easily shared by the local group provided that your system administrator is willing to put a copy of your package in the `$(MATVECHOME)/packages/contrib` directory.

As a matter of fact, lots of Matvec standard functions such as `A.sign()` are macros. They are called the Matvec default macro packages. The differences between a default macro package and a contribution package are

- the former is automatically loaded into memory whenever Matvec is launched.
- the former is residing in the `$(MATVECHOME)/packages/default` directory.
- the former is used much more often than the latter.

### 9.1 Demo Package

The macro package `demo` is designed for beginners to have a taste of Matvec system. It provides two functions: `demo()` and `demo_src()`. To use one of these two functions, you first have to load the whole package into computer memory using Matvec function `package( "demo" )`.

### 9.1.1 demo() function

Here is the macro script of demo() function:

```
function demo(program_name)
{
  #
  # run a demo program in $(MATVECHOME)/examples/interface/txt
  #
  # Tianlin Wang at UIUC, Sat Jan 28 16:13:05 CST 1995
  #

  local(retval,filename);
  if (program_name.class() == "String") {
    filename = getenv("MATVECHOME") + "/examples/interface/txt/";
    filename += program_name;
    retval = input(filename);
  }
  else {
    error("demo(program_name): program_name must be a string");
    retval = 0;
  }
  return retval;
}
```

There are a number of demo examples coming with the Matvec standard distribution. Usually, these examples are residing in MATVECHOME/examples/interface/txt directory, where MATVECHOME can be obtained using Matvec function getenv("MATVECHOME"). These demo examples are listed and briefly described below:

**hello:** greetings to every body.

**gianola:** matrix version of BLUP.

**try:** variance component estimation based on a five-animal pedigree

**VG:** single trait variance component estimation.

**susan:** single trait BLUP.

For examples

```
> package("demo");
> demo("try")
      .matvec.799959823.0
      .matvec.799959823.1
iteration  sigma_1 ..... sigma_e  log_likelihood
0:         1         2      -6.10457
1:    1.74667    2.87176    -6.48199
2:    1.65664    2.91128    -4.73543
3:    2.23916    2.68895    -5.76399
4:    2.22348    2.6937   -5.76309
5:    2.21558    2.69613    -5.76264
6:    2.21156    2.69737    -5.76241
7:    2.20951    2.69801    -5.76229
8:    2.20846    2.69833    -5.76223
9:    2.20792    2.6985   -5.7622
```

```

10:      2.20765      2.69859      -5.76218
11:      2.20751      2.69863      -5.76217

      some extra information in the model
-----
variance for animal =

      Col 1
Row 1      2.20751
residual variance =

      Col 1
Row 1      2.69863

MME dimension      : 8
non-zeros in MME: 23

      basic statistics for dependent variables
-----
      trait-name      n      mean      std
      y      6      8.33333      2.06559
-----

```

### 9.1.2 demo\_src() function

`demo_src("<program_name>")` is used to displays the source code of demo program "<program\_name>"; whereas `demo_src()` displays the source code of the demo macro package itself.

Here is the macro script of `demo_src()` function:

```

function demo_src(program_name)
{
  #
  # list a demo program in $(MATVECHOME)/examples/interface/txt
  #
  # Tianlin Wang at UIUC, Sat Jan 28 16:13:05 CST 1995
  #

  local(retval,shellcmd,filename);
  if (program_name.class() == "String") {
    filename = getenv("MATVECHOME") + "/examples/interface/txt/";
    filename += program_name;
    shellcmd = PAGER + " " + filename;
    retval = system(shellcmd);
  }
  else {
    error("demo_src(program_name): program_name must be a string");
    retval = 0;
  }
  return retval;
}

```

For examples

```

> package("demo");
> demo_src("try")

```

```
##### try #####
data_file_name = RawData{
  A2    1    1.0    6.0
  A2    1    1.0    8.0
  A3    1    1.0    .
  A3    1    1.0    8.0
  A4    .    0.0    5.0
  A4    2    2.0    7.0
  A5    2    2.0    9.0
  A5    2    2.0   12.0
}

ped_file_name = RawData {
  A1      .      .
  A2      .      .
  A3      A1     A2
  A4      .      A2
  A5      .      A3
}

D = Data();
D.input(data_file_name,"animal$ herd _skip y");

P = Pedigree();
P.input(ped_file_name);

M = Model("y = intercept herd animal");
M.variance("residual",2.0);
M.variance("animal",P,1.0);
M.fitdata(D);

M.vce_emreml(); // correct answer: sigma_e = 2.69863, sigma_a = 2.20751
```

## 9.2 Prime Number Package

In the prime number package, there are three interesting functions related to prime number.

### 9.2.1 prime(n)

It returns the first n prime numbers.

### 9.2.2 prime\_next(n)

It returns the prime number next to n. If n is the prime number, then it simply returns n itself.

### 9.2.3 prime\_less(n)

It returns a vector containing all prime numbers less or equal to the argument n.

### 9.2.4 examples

```
> package("prime");
> prime(4)
```

```

      i = 1      i = 2      i = 3      i = 4
        2        3        5        7
> prime_next(4)
5
> prime_next(888)
907
> prime_less(8)

      i = 1      i = 2      i = 3      i = 4
        2        3        5        7

```

## 9.3 Special Matrix Package

The macro package `special_matrix` contains following special matrices: `hilb`, `invhilb`, `hankel`, `vander`, `hadamard`, `pascal`, `toeplitz`.

### 9.3.1 hilb

`hilb(n)` creates the Hilbert matrix of order `n`. The `i, j` element  $H(i,j)$  of a Hilbert matrix is defined as  $H(i,j) = 1 / (i + j - 1)$ . For example,

```

> package("special_matrix");
> hilb(3)

```

	Col 1	Col 2	Col 3
Row 1	1	0.5	0.333333
Row 2	0.5	0.333333	0.25
Row 3	0.333333	0.25	0.2

### 9.3.2 invhilb

`invhilb(n)` returns the inverse of the Hilbert matrix of order `n`. This is the exact results. See and compare `invhilb(n)` with `hilb(n).inv()`. For example,

```

> package("special_matrix");
> invhilb(3)

```

	Col 1	Col 2	Col 3
Row 1	9	-36	30
Row 2	-36	192	-180
Row 3	30	-180	180

### 9.3.3 hankel

`hankel(C)` returns a square Hankel matrix whose first column is `C` and whose elements are zero below the first anti-diagonal; whereas `hankel(C,R)` returns a Hankel matrix whose first column is `C` and whose last row is `R`. Hankel matrices are symmetric, constant across the anti-diagonals, and have elements  $H(i,j) = R(i+j-1)$ . For example,

```

> package("special_matrix");
> hankel([1,2,3])

```

	Col 1	Col 2	Col 3
--	-------	-------	-------

```

      Row 1      1      2      3
      Row 2      2      3      0
      Row 3      3      0      0
> hankel([1,2,3],[3,4,5])

```

```

      Col 1      Col 2      Col 3
      Row 1      1      2      3
      Row 2      2      3      4
      Row 3      3      4      5

```

### 9.3.4 vander

vander(C) returns the Vandermonde matrix whose second to last column is C. The j-th column of a Vandermonde matrix is given by  $A(*,j) = C^{(n-j)}$ . For example,

```

> package("special_matrix");
> vander([1,2,3])

```

```

      Col 1      Col 2      Col 3
      Row 1      1      1      1
      Row 2      4      2      1
      Row 3      9      3      1

```

### 9.3.5 hadamard

hadamard(k) return the Hadamard matrix of order  $n = 2^k$ .

not available

### 9.3.6 pascal

pascal(n) returns the Pascal matrix of order n: a symmetric positive definite matrix with integer entries, made up from Pascal's triangle. For example,

```

> package("special_matrix");
> pascal(3)

```

```

      Col 1      Col 2      Col 3
      Row 1      1      1      1
      Row 2      1      2      3
      Row 3      1      3      6

```

### 9.3.7 toeplitz

toeplitz(C) returns a symmetric (or Hermitian) Toeplitz matrix; whereas toeplitz(C,R) returns a non-symmetric Toeplitz matrix having C as its first column and R as its first row. For example,

```

> package("special_matrix");
> toeplitz([1,2,3])

```

```

      Col 1      Col 2      Col 3
      Row 1      1      2      3
      Row 2      2      1      2
      Row 3      3      2      1

```



```
> toeplitz([1,2,3],[1,4,5])
```

	Col 1	Col 2	Col 3
Row 1	1	4	5
Row 2	2	1	4
Row 3	3	2	1

## 9.4 Linear Programming Package

not available yet



# Chapter 10

## Linear Model Analyses

### 10.1 Linear Regression

...

### 10.2 Best Linear Unbiased Estimation (BLUE)

...

### 10.3 Best Linear Unbiased Prediction (BLUP)

#### 10.3.1 Data

Suppose we have a five-animal pedigree stored in an ASCII file called `try.ped`:

```
// a animal pedigree, created by Tianlin Wang
// note that unknown parents can also be represented by 0
// individual father mother
A1      .      .
A2      .      .
A3      A1     A2
A4      .      A2
A5      .      A3
```

Then, the eight-line records from four animals were obtained, and stored in another ASCII file called `try.dat`:

```
// a small test data, created by Tianlin Wang
// animal herd wt y
A2  1  1.0  6.0
A2  1  1.0  8.0
A3  1  1.0  .
A3  1  1.0  8.0
A4  .  0.0  5.0
A4  2  2.0  7.0
A5  2  2.0  9.0
A5  2  2.0  12.0
```

There are several Matvec features shown in these two files:

- either the pedigree file or the raw data file can contain comments which is started from the double backslash. This could be very handy when you want to write down a note about your data. Comments and empty lines will be quietly ignored by Matvec.
- missing values must be represented by a dot(.) with an exception that unknown parents in the pedigree file can also be represented by 0 (the cost is that we can't use 0 as a legitimate animal identification any more). Missing values can appear anywhere in both files. Matvec will automatically and appropriately handle missing values for you.
- animal or herd identification can be either a string or a number.

In the following several sections, we will demonstrate how to analyze the above data using linear model methodology.

### 10.3.2 Model

A single trait animal model is assumed:

$$y_i = \mu + h_i + a_i + e_i$$

### 10.3.3 Obtain BLUP using matrix algebra

First of all, we write down a script using matrix tools and functions provided by Matvec. You can use any of your favorable ASCII editor, and save the file as `try1` in your current working directory.

```
A = [1.0, 0.0, 0.5, 0.0, 0.25;
      0.0, 1.0, 0.5, 0.5, 0.25;
      0.5, 0.5, 1.0, 0.25, 0.5;
      0.0, 0.5, 0.25, 1.0, 0.125;
      0.25, 0.25, 0.5, 0.125, 1.0]; // A = relationship matrix

X = [1.0, 1.0, 0.0;
      1.0, 1.0, 0.0;
      1.0, 1.0, 0.0;
      1.0, 0.0, 1.0;
      1.0, 0.0, 1.0;
      1.0, 0.0, 1.0];

Z = [0.0, 1.0, 0.0, 0.0, 0.0;
      0.0, 1.0, 0.0, 0.0, 0.0;
      0.0, 0.0, 1.0, 0.0, 0.0;
      0.0, 0.0, 0.0, 1.0, 0.0;
      0.0, 0.0, 0.0, 0.0, 1.0;
      0.0, 0.0, 0.0, 0.0, 1.0];

y = [6.0;
      8.0;
      8.0;
      7.0;
      9.0;
      12.0];

XPX = X'*X;
XPZ = X'*Z;
ZPZ = Z'*Z;
```

```

XPy = X'*y;
ZPy = Z'*y;
rhs = [XPy;
       ZPy];

sigma_e = 2.0;
sigma_a = 1.0;
r = sigma_e/sigma_a;
G = A*sigma_a;

MME = [XPX,  XPZ;
       XPZ', ZPZ + A.inv()*r];

blup = MME.ginv()*rhs

```

Now, at the Matvec prompt, simply type

```

> input try1
blup =

```

	Col 1
Row 1	5.51000
Row 2	1.91000
Row 3	3.60000
Row 4	0.260000
Row 5	-0.260000
Row 6	0.260000
Row 7	-0.670000
Row 8	0.670000

The above script written in Matvec language is the exactly what Professor Gianola told me to do when I was taking his linear model course. Step by step, I have clearly shown you what I was doing in the above script. Now I will do the same job in an encapsulated but efficient way using Matvec higher level functions.

### 10.3.4 Obtain BLUP using Matvec higher level functions

M.blup() returns the blup as a vector if M is a *Model* object. I wrote down the following Matvec script and saved it in an ASCII file called try2:

```

P = Pedigree();
P.input("try.ped");

D = Data();
D.input("try.dat","animal\$ herd _skip y");

M = Model();
M.equation("y = intercept + herd + animal");
M.variance("residual",2);
M.variance("animal",P,1);
M.fitdata(D);

M.blup()

```

There are two Matvec features I want to mention here:

- In statement

```
D.input("try.dat","animal\$ herd _skip y");
```

the dollar sign (\\$) after animal is used to indicate that this data-column is string column. The `_skip` is one of reserved data-column name meaning to *skip* or ignore the column.

- In statement

```
M.equation("y = intercept + herd + animal");
```

the `intercept` is a reserved model-term meaning an intercept or a grand-mean. The `+` between model terms is optional. The interaction term or the nested term can be expressed in the same way as SAS. For instance, `A*B` and `x(A)`. The above statement about model can also be re-written as

```
M.equation("y = intercept +",
           "herd + animal");
```

This should give you enough hint about how to break down a model with dozens of terms.

If you add a semicolon(;) after statement `M.blup()`, then BLUP solution would not be displayed on screen.

Now, at the Matvec prompt, I type

```
> input try2
  i = 1      i = 2      i = 3      i = 4      i = 5
  7.42        0        1.69        0.26       -0.26

  i = 6      i = 7      i = 8
  0.26      -0.67      0.67
```

If you want a more descriptive output for BLUP, then use statement `M.save("try.out")` right after `M.blup()`. The following could be the possible contents in "try.out":

```

      some extra information in the model
-----
variance for animal =

          Col 1
      Row 1      1
residual variance =

          Col 1
      Row 1      2

MME dimension      : 8
non-zeros in MME: 23

      basic statistics for dependent variables
-----
trait-name      n      mean      std
      y          6      8.33333      2.06559
-----

      BLUP (BLUE, Posterior_Mean)
-----
```

Term	Trait 1
intercept	y

1	7.42
---	------

Term	Trait 1
herd	y

1	0
2	1.69

Term	Trait 1
animal	y

A1	0.26
A2	-0.26
A3	0.26
A4	-0.67
A5	0.67

Note that because of different g-inverse algorithms, the estimates of herds effects seems quite different from the two above programs. However, the value of the estimable function h1-h2 is identical. Of course, the BLUP values for animal additive effects are identical, too.

There are several optional arguments for M.blup() function to let user to choice appropriate solver, maximum iterations allowed, and stop criterion. See blup entry in the reference manual for more details.

### 10.3.5 Multi-trait and multi-model example

```
P = Pedigree();
Ped.input("large.ped");

D = Data();
D.input("large.dat",
       "animal\$ testdate line herd sex weight bf dg_test fce dg_farm");

M = Model("bf      = weight(line) line      animal",
          "dg_test =          line testdate animal",
          " fce     =          line testdate animal",
          "dg_farm = herd sex   line testdate animal");

M.covariate("weight");
M.variance("residual",
          2.8,   10.7,   0.0182,  0.5,
          10.7,  4160.0, -3.64,   1050.0,
          0.0182, -3.64,  0.026,  -1.3,
          0.5,   1050.0, -1.3,   1840.0);

M.variance("animal",P,
          1.2,   7.7,   0.003,  3.32,
          7.7,   2240.0, -1.68,  470.0,
          0.003, -1.68,  0.014, -0.72,
          3.32,  470.0,  -0.72,  410.0);
```

```
M.fitdata(D);
M.blup();
M.save("mme.out");
```

## 10.4 Linear Estimation

`M.estimate(Kp)` returns  $\text{BLUE}(K'b)$ , `M.covmat(Kp)` returns  $\text{Var}(K'\hat{b}-K'b)$ , and `M.label(i)` returns the label of model effect  $i$  where `Kp` means  $K'$ . For example, the following Matvec script is saved in a file called `try3`.

```
P = Pedigree();
P.input("try.ped");

D = Data();
D.input("try.dat","animal\$ herd _skip y");

M = Model();
M.equation("y = intercept herd animal");
M.variance("residual",2);
M.variance("animal",P,1);
M.fitdata(D);

Kp=[1 1 0
     1 0 1
     0 1 ,-1];
herd=M.estimate(Kp);
var=M.covmat(Kp);
for(i=1;i<=3;i++) {
  string("Effect ",i," ",M.label(i))
}
string("    Herd 1 ",herd(1)," +- ",var(1,1).sqrt(),"\n")
string("    Herd 2 ",herd(2)," +- ",var(2,2).sqrt(),"\n")
string("Herd 1 - 2 ",herd(3)," +- ",var(3,3).sqrt(),"\n")
```

Now, at the Matvec prompt, I type

```
> input try3
    Effect 1 intercept:1
    Effect 2 herd:1
    Effect 3 herd:2
        Herd 1 7.42 +- 1.19861

        Herd 2 9.11 +- 1.12213

    Herd 1 - 2 -1.69 +- 1.3997
```

## 10.5 Linear Hypothesis Test

`M.contrast(Kp,m)` is doing a linear hypothesis test that  $H_0: K'b = m$ . Vector  $m$  is optional, the default is a vector of 0's. For example, the following Matvec script is saved in a file called `try4`.

```
P = Pedigree();
P.input("try.ped");
```



```

D = Data();
D.input("try.dat","animal\$ herd _skip y");

M = Model();
M.equation("y = intercept herd animal");
M.variance("residual",2);
M.variance("animal",P,1);
M.fitdata(D);

Kp = [0,1,-1];
M.contrast(Kp)

```

Now, at the Matvec prompt, I type

```
> input try4
```

```

RESULTS FROM CONTRAST(S)
-----
Contrast MME_addr      K_coef      Raw_data_code
-----
1          2          1   herd:1
1          3         -1   herd:2
      estimated value (K'b-M) = -1.69 +- 1.3997
      Prob(|t| > 1.2074) = 0.227421 (p_value)
-----
0.227278

```

The last two statements in the above Matvec script can be replaced by a single statement:

```
M.contrast("herd",[1,-1])
```

Any composite hypothesis tests require multi-row matrix Kp. For instance, the test  $H_0: h_1 = h_2 = 0$  can be accomplished in Matvec as below:

```
M.contrast("herd",[1,0; 0,1])
```

## 10.6 Least Squares Means (lsmeans)

The least-squares means (lsmeans), also called population marginal means, are the expected values of class or subclass means that you would expect for a balanced design involving the class variable with all covariates at their mean level. A least-squares mean for a given level of a given model-term may not be estimable.

M.lsmeans("<termname-list>") outputs the least-squares means for each terms in termname-list. Here is the example saved in a file try5

```

D = Data();
D.input("try.dat","animal\$ herd _skip y");

P = Pedigree();
P.input("try.ped");

M = Model();
M.equation("y = intercept herd animal");
M.variance("residual",2.0);
M.variance("animal",P,1.0);
M.fitdata(D);

M.lsmeans("herd");

```

Now, at the Matvec prompt, I type

```
> input try5
```

LEAST SQUARE MEANS				
herd	lsmean	sqrt(Var(kb))	p_value	estimability
1	7.42	1.4594	0.00705964	
2	9.11	1.36628	0.00262883	

```
note 1: estimability = Max(mme*ginv(mme)*k - k);
        if it is non-zero, lsmean may or may not be estimable.
note 2: sqrt(Var(kb)) is also called STD ERR (see SAS)
```

## 10.7 Variance Components Estimation (VCE)

There are two member functions of a model object for variance components estimation: `M.vce_emreml()` and `M.vce_dfreml()`.

`M.vce_emreml(maxiter, interactive, stopval)` estimates the variance components for random effects using EM-REML algorithm. Three arguments are optional. The first argument `maxiter` is the maximum number of iterations allowed with the default value 1000; if you are running Matvec interactively, it allows you to add more iterations at run-time, thus a recommended `maxiter` value is 20 in order to monitor increment of restricted log-likelihood. The second argument `interactive` is the switch for interactive mode or non-interactive with the default value 1. The third argument `stopval` is the stop criterion with the default value 1.0e-5.

`M.vce_dfreml("df_method", maxiter, stopval)` estimates the variance components for random effects using DF-REML algorithm. Three arguments are optional. The first argument "`<df_method>`" is the name of derivative-free based maximization maximum algorithm with default name "`powell`"; the second `maxiter` is the maximum number of iterations allowed with default value 500000; the third `stopval` is the stop criterion with default value 1.0e-5.

Both member functions return the BLUP solution which used the latest estimated variance components. Therefore, a semicolon (;) should always be used to end the functions to suppress printing of the length BLUP solution.

For example, the following Matvec script is saved in a file called `try6`.

```
D = Data();
D.input("try.dat", "animal\$ herd _skip y");

P = Pedigree();
P.input("try.ped");

M = Model();
M.equation("y = intercept + herd + animal");
M.variance("residual", 2.0); // initial variance value
M.variance("animal", P, 1.0); // initial variance value
M.fitdata(D);

M.vce_emreml(); // correct answer: sigma_e = 2.69863,
                //                  sigma_a = 2.20751
```

Now, at the Matvec prompt, I type

```
> input try6
```

iteration	sigma_1	.....	sigma_e	log_likelihood
0:	1		2	-6.10457331

1:	1.74667	2.87176	-6.481992428
2:	1.65664	2.91128	-4.735431018
3:	2.23916	2.68895	-5.763994369
4:	2.22348	2.6937	-5.763092324
5:	2.21558	2.69613	-5.762636885
6:	2.21156	2.69737	-5.762405211
7:	2.20951	2.69801	-5.762286907
8:	2.20846	2.69833	-5.762226375
9:	2.20792	2.6985	-5.762195371
10:	2.20765	2.69859	-5.762179483
11:	2.20751	2.69863	-5.762171339

some extra information in the model

-----  
variance for animal =

	Col 1
Row 1	2.20751

residual variance =

	Col 1
Row 1	2.69863

MME dimension : 8  
non-zeros in MME: 23

basic statistics for dependent variables

trait-name	n	mean	std
y	6	8.33333	2.06559

-----



## Chapter 11

# Generalized Linear Mixed Model Analyses

### 11.1 Generalized Linear Model

A study was conducted to examine the dose response of two alternative formulations of an antibiotic. A series of concentrations (1, 2, 4, 8, 16, 32, 64) of the two formulations (A and B) were examined. Ten tubes of each formulation were prepared at each concentration and the proportion of tubes with bacterial growth was recorded. The data for this experiment are presented in Table 11.1. The dose response curves for the two formulations are presented in Figure 11.1.

The data for this study are stored in the file `dose.dat`:

```
A 1 10 10 1.0 0
A 2 10 10 1.0 1
A 4 4 10 .4 2
A 8 1 10 .1 3
A 16 0 10 .0 4
A 32 0 10 .0 5
A 64 0 10 .0 6
B 1 10 10 1.0 0
B 2 9 10 .9 1
```

As with a linear model the data are read into a Data object:

```
D=Data();
D.input("dose.dat","Form $ dose pos n perc_pos log2dose");
```

Table 11.1: Bacterial growth in the presence of two different antibiotic formulations.

Dose	Formulation A		Formulation B	
	Num. Pos.	Prop. Pos.	Num. Pos.	Prop. Pos.
1	10	1.0	10	1.0
2	10	1.0	9	0.9
4	4	0.4	10	1.0
8	1	0.1	4	0.4
16	0	0.0	1	0.1
32	0	0.0	0	0.0
64	0	0.0	1	0.1

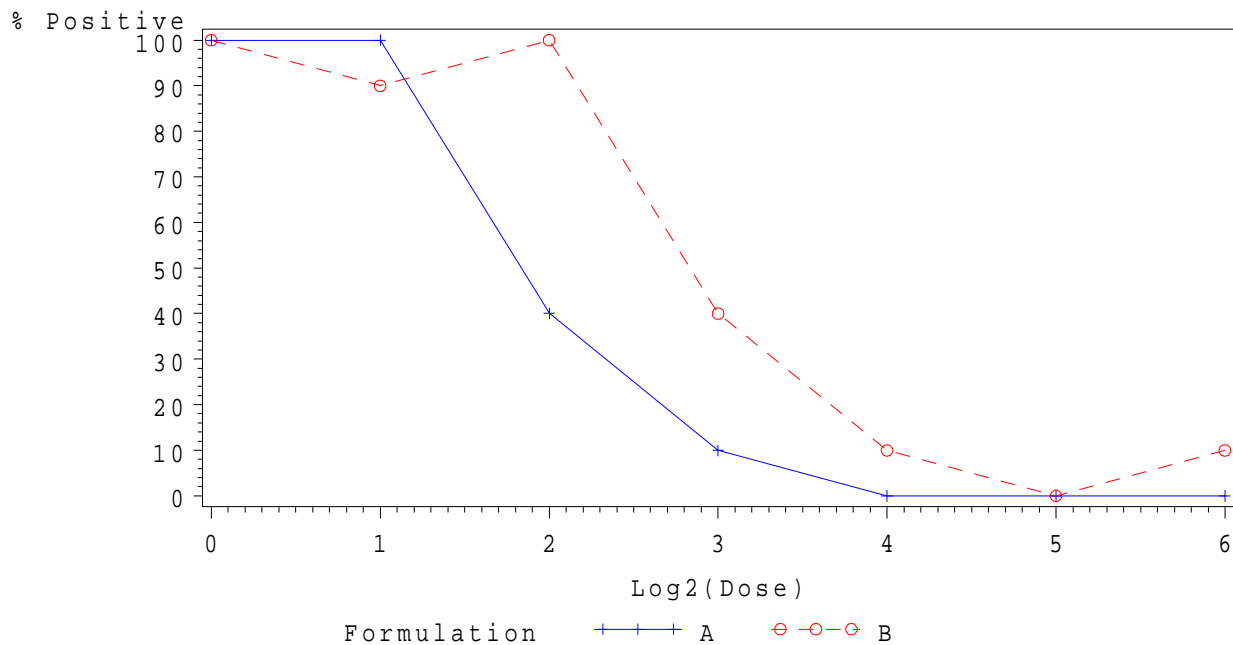


Figure 11.1: Dose response curves for two different formulations of an antibiotic.

### 11.1.1 Model Specification

The specification of a generalized linear model consists of descriptors for the linear predictor and the link function. The linear predictor for our example is:

$$\eta_{ij} = \mu + F_i + b x_j + b_i x_j$$

- $\mu$ : intercept
- $F_i$ : main effect of formulation  $i$  at  $x = \log_2(\text{Dose}) = 0$
- $b$ : linear effect of  $\log_2(\text{Dose})$
- $b_i$ : linear interaction between formulation and  $\log_2(\text{Dose})$ .

In Matvec this linear predictor is denoted by:

```
M=Model(); // Create a Model object.
M.equation("perc_pos=intercept Form log2dose Form*log2dose");
// Linear predictor  $\eta_{ij} = \mu + F_i + b x_j + b_i x_j$ 
M.covariate("log2dose");
M.weight("n");
```

which is the same as a linear model.

Next the logit link function is specified:

```
M.link("logit",0)
// Logit link, scale parameter is not estimated.
M.fitdata(D);
// Use Data object D
```

`M.link('fn',nvc)` specifies both the link function, `'fn'`, and the number of over-dispersion parameters, `nvc`, to estimate. Table 11.2 contains a number link functions that are currently available.

Table 11.2: Available link functions

Function	Distribution	Link	Inverse Link	$v(\mu)$
<b>normal</b>	Multi. Normal	Identity	$\eta$	$\mathbf{R} = \{\sigma_{ij}\}$
<b>normal log</b>	Multi. Normal	Identity	$\eta$	$\mathbf{R} = \exp(\{\sigma_{ij}\})$
<b>logit</b>	Binomial/ $n$	Logit	$e^\eta / (1 + e^\eta)$	$\mu(1 - \mu)/n$
<b>probit</b>		Probit	$\Phi(\eta)$	
<b>cloglog</b>		Complementary log-log	$1 - \exp(-\exp(\eta))$	
<b>thresh</b>	Multinomial	Threshold	$\Pr(\eta + \tau_{i-1} < Z \leq \eta + \tau_i)$	$\text{Diag}(\boldsymbol{\mu}) - \boldsymbol{\mu}\boldsymbol{\mu}'$
<b>poisson</b>	Poisson	Log	$e^\eta$	$\mu$
<b>weibull</b>	Weibull			
<b>richards</b>	Richards Growth Function		$\eta_1(1 \pm e^{-\eta_3(t-t\eta_2)})^{\eta_4}$	$\sigma^2$

### 11.1.2 Maximum Likelihood Estimates

For generalized linear mixed models, estimates of the fixed effects are obtained by iteratively solving

$$(\mathbf{X}'\mathbf{H}'\mathbf{R}^{-1}\mathbf{H}\mathbf{X})\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{H}'\mathbf{R}^{-1}\mathbf{y}^*) \quad (11.1)$$

where

$$\begin{aligned} \mathbf{H} &= \frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{\eta}'} \\ \mathbf{R} &= \text{Var}(\mathbf{y}) \\ \mathbf{y}^* &= \mathbf{y} - \boldsymbol{\mu} + \mathbf{H}\boldsymbol{\eta}. \end{aligned}$$

The `M.glim()` member function can be used to iteratively solve (11.1) and returns a vector of the estimated effects. With no arguments `M.glim()` performs 10 iterations. In this case, we will perform 20 iterations by using

```
M.glim(20);
```

Convergence can be checked by comparing the current log likelihood

```
"Log Like " L=M.log_like()
```

with the log likelihood after a few additional iterations

```
bhat=M.glim(5);
```

```
"Change in Log Like " (M.log_like()-L)
```

where the vector `bhat` contains the estimated effects. The resulting output:

```
Log Like
-32.1018
Change in Log Like
0
```

indicating that the algorithm did in fact converge.

### 11.1.3 Estimation and Testing

As with linear models estimates of  $\mathbf{K}'\boldsymbol{\beta}$  can be obtained using `M.estimate(Kp)`, where `Kp` is  $\mathbf{K}'$ . For example to estimate the effect of formulation:

```
Kp=[0 1, -1 0 0 0
     0 0  0 0 1 , -1]
"Estimate ",est=M.estimate(Kp)
"Kp*bhat ",Kp*bhat
```

with the result:

```
Estimate
1.51543
-1.51994
Kp*bhat
1.51543
-1.51994
```

Wald tests and standard errors can be obtained using the `M.contrast(Kp)` member function. For example, to test for a formulation effect:

```
"\nFormulation Effect p-value="+string(pval=M.contrast(Kp))
```

with the result:

```
RESULTS FROM CONTRAST(S)
-----
Contrast MME_addr    K_coef    Raw_data_code
-----
1          2          1    Form:A
1          3         -1    Form:B
      estimated value (K'b-M) = 1.51543 +- 2.31358
      Prob(|t| > 0.655017) = 0.512532 (p_value)

2          5          1    Form*log2dose:A*log2dose
2          6         -1    Form*log2dose:B*log2dose
      estimated value (K'b-M) = -1.51994 +- 1.03644
      Prob(|t| > 1.46651) = 0.142666 (p_value)
-----
joint hypothesis test H: K'b = M
Prob(F > 3.67092) = 0.0254529 (p_value)
```

```
Formulation Effect p-value=0.0254529
```

A likelihood ratio test can also be performed by fitting a reduced model with the restriction  $\mathbf{K}'\boldsymbol{\beta} = 0$  and comparing the log-likelihoods of the full and reduced models. For example, to calculate the likelihood ratio test for a formulation effect:

```
M.glim(20,Kp);
string("\nReduced Log Like=",RL=M.log_like())
Chi2=StatDist("ChiSquare",2);
string("\nLRT =",TS=2*(L-RL)," \n", "P-value=",1-Chi2.cdf(TS))
```

with the result:

```
Reduced Log Like=-38.2602
```

```
LRT =12.3139
P-value=0.00211866
```



### 11.1.4 Threshold Model

Matvec can also be used to fit threshold models based on either an underlying normal,  $\Pr(Z \leq x) = \Phi(x)$ , or logistic,  $\Pr(Z \leq x) = e^x / (1 + e^x)$  distribution. Suppose data has been collected on calving difficulty to examine the effect of gender of the calf on calving difficulty. The data are given in Table 11.3.

Table 11.3: Calving difficulty data.

Sex	Calving Difficulty		
	Easy	Moderate	Difficult
Male	169	16	15
Female	161	23	16

The threshold link function in Matvec requires that ordinal data with  $c$  categories be coded as  $0, 1, \dots, c - 1$ . The probability that an observation falls in category  $i$  is given by

$$\Pr(Y = i) = \Pr(\tau_{i-1} + \eta < Z \leq \tau_i + \eta)$$

where  $\tau_{-1} = -\infty$ ,  $\tau_0 = 0$  and  $\tau_c = \infty$ . The data for this study are stored in the file `calve.dat`:

```
Male 0 169
Male 1 16
Male 2 15
Female 0 161
Female 1 23
Female 2 16
```

and are read using:

```
D=Data();
D.input("calve.dat","Sex $ Score n");
```

For this example, the linear predictor for observation  $j$  will include an intercept and sex main effect. The threshold between moderate (1) and difficult (2) will only include an intercept. The model is specified by:

```
M=Model();
M.equation("Score= intercept Sex,Score=intercept");
M.link("thresh",0)
M.param(1);
M.weight("n");
M.fitdata(D);
```

The `M.param(1)` indicates that a probit threshold model will be used. Alternatively `M.param(0)` can be used to fit a logistic threshold model. The analysis is completed using:

```
M.glim(20)
M.save("calve.out");
M.contrast("Sex",[1 0 ,-1 0])
```

Resulting in the following:

```
Threshold Link Function
Parameters: Probit/Logit (scalar)) 1=probit 0=logit
Terms:      Eta1, Thresh1, Thresh2, ...
y 0..numtraits
0
0.48819
```

```
1.00175
0
0.872275
0
```

#### RESULTS FROM CONTRAST(S)

```
-----
Contrast MME_addr    K_coef    Raw_data_code
-----
1          3          1    Score:Sex:Male
1          5         -1    Score:Sex:Female
      estimated value (K'b-M) = 0.129475 +- 0.144313
      Prob(|t| > 0.897178) = 0.369732 (p_value)
-----
0.369624
```

The results indicating that the threshold for more difficult births is raised by 0.13 for cows giving birth to male calves. However, the difference between cows giving birth to male calves and cows giving birth to female calves was not significant (p-value=.37). The model fitting information is summarized in the file `calve.out`:

#### some extra information in the model

```
-----
residual variance =
```

```
MME dimension    : 6
non-zeros in MME: 11
```

#### basic statistics for dependent variables

```
-----
      trait-name      n      mean      std
Score                6      1.00000    0.894427
Score                6      1.00000    0.894427
-----
```

#### BLUP (BLUE, Posterior\_Mean)

```
-----
      Term      Trait 1      Trait 2
intercept Score Score
      1      0.00000      0.488190

      Term      Trait 1      Trait 2
Sex Score Score
      Male      1.00175
      Female     0.872275
```

### 11.1.5 Survival analysis

Matvec can be used to analyze survival data using a Weibull survival model. Table 11.4 contains remission times for cancer patients on two types of drug therapies. Figure 11.2 contains the survival functions for the groups of patients. The data file `cancer.dat` contains the remission times for the 42 patients:

```
A 0 1
A 1 1
A 1 1
```

Table 11.4: Remission times of cancer patients.

Weeks (10)	A	B
0	011234558	00001123334556779
1	002356	137
2	014	0
3	139	

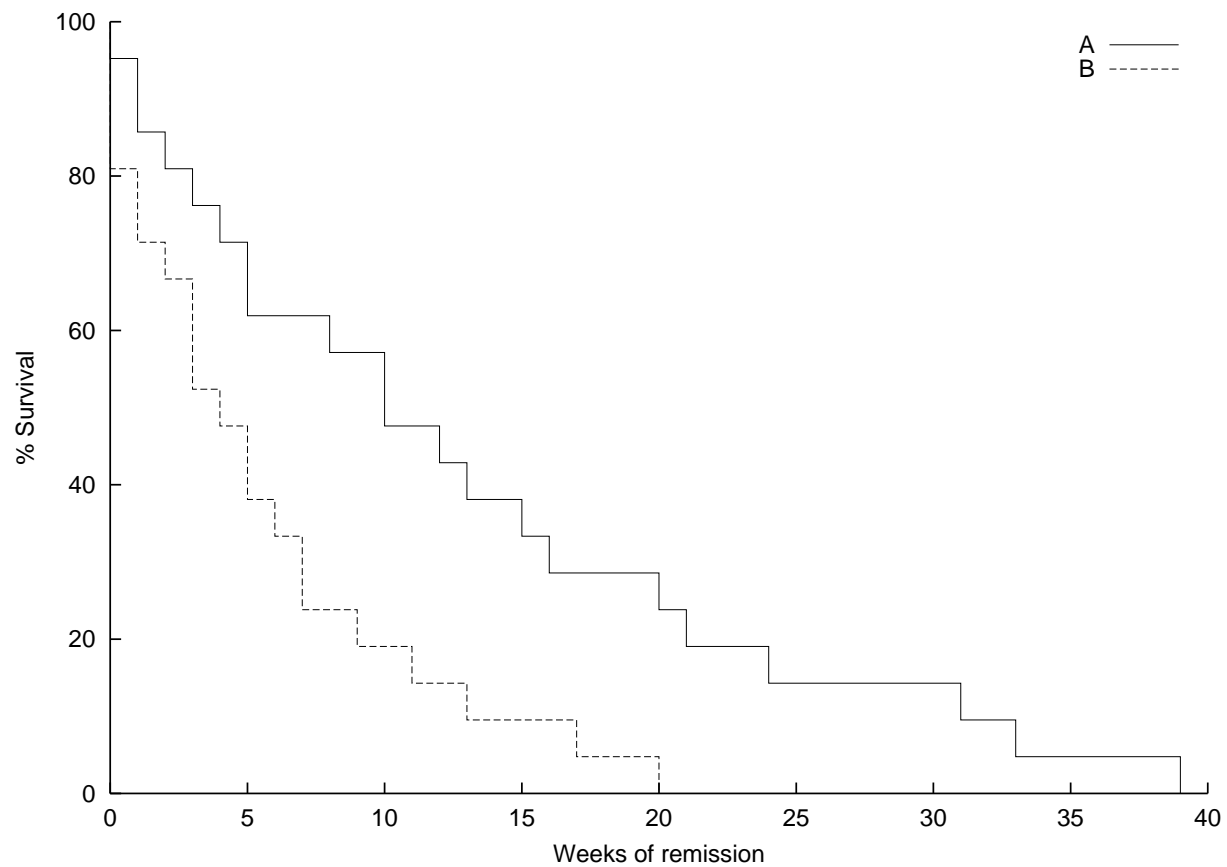


Figure 11.2: Cancer remission times for two drug therapies.

```

:
B 13 1
B 17 1
B 20 1

```

The third column contains a code to indicate if the data are right censored (1:uncensored and 0:censored). Because none of the records were censored, all the censor codes are one in this example. Like the threshold model, the Weibull is a multivariate generalized linear model. Model equations will be needed for both a risk factor  $\eta$  and a rate parameter  $\rho$ . Typically the rate parameter, which describes the shape of the survival function, is assumed to be the same for everyone in the population and only includes an intercept term. The risk factor will include both an intercept and an effect for drug therapy.

The Weibull survival model for this example can be obtained from:

```

D=Data();
D.input("cancer.dat","therapy $ week censor");
M=Model();
M.equation("week=intercept therapy,week=intercept");
M.link("weibull",0)
M.param(0,"week","censor",.5);
M.init(-1,1);
M.fitdata(D);
M.glim();
M.save("cancer.out");

```

The `M.param()` is used to specify four additional model fitting parameters needed by the `weibull` link function. The first parameter is 1 if the rate parameter is to remain fixed and 0 if the rate parameter is to be estimated. The second parameter, is the time variable in the data set. The third parameter, is the censor code variable in the data set. The fourth parameter is the time to recode any zero times to in the data set. The `M.init()` is used to specify initial values for both the risk factor and the rate parameter. If `M.init()` was not included the initial values for the risk factor and rate factor would be zero.

A Wald test for testing for a therapy effect is generated by:

```

"\n Therapy effect\n"
M.contrast("therapy",[1,0,-1,0]);

```

To test the hypothesis that the rate parameter was 1 is generated by:

```

"\n Rate parameter\n"
M.contrast("intercept",[0 1],[1]);

```

The output from the program is

```

      Weibull Link Function
Parameters: FixRho (scalar), Time variable (string),
           Censor Variable (string) 1=uncen 0=cen,
           Zero Value (scalar)
Terms:      Eta, Rho

```

Therapy effect

#### RESULTS FROM CONTRAST(S)

Contrast	MME_addr	K_coef	Raw_data_code
1	3	1	week:therapy:A

```

1          5          -1   week:therapy:B
      estimated value (K'b-M) = -0.86038 +- 0.323848
      Prob(|t| > 2.65675) = 0.00795264 (p_value)
-----

```

Rate parameter

#### RESULTS FROM CONTRAST(S)

```

-----
Contrast MME_addr    K_coef    Raw_data_code
-----
1          2          1   week:intercept:1
      estimated value (K'b-M) = 0.0367026 +- 0.128135
      Prob(|t| > 0.286437) = 0.774573 (p_value)
-----

```

Finding a significantly reduced risk with drug therapy A and that the survival function did not differ significantly from a simple exponential model.

### 11.1.6 Growth Curves

Matvec can be used analyze growth curve data using the Richards family of growth curves. The Richards growth curve is a 4 parameter nonlinear function. The parameterization of Richards growth function used is

$$A(1 \pm e^{-b(t-t^*)})^m$$

where the growth parameters are  $A$  the asymptotic weight,  $t^*$  a location parameter,  $b$  a rate parameters, and  $m$  a shape parameter. The  $\pm$  in the above equation is  $+$  when  $m < 0$  and  $-$  when  $m > 0$ . The Richards growth function is very flexible. However, it can be very difficult to work with. Typically a researcher will look at special cases by fixing the value of the shape parameter  $m$ . With  $m = -1$  the Richards function is the logistic growth function and with  $m = 1$  the Richards function is Brody's growth function.

The data in Table 11.5 will be used to illustrate fitting both a logistic and Richards growth functions. For the logistic there are three linear predictors to be modeled each of which will include an intercept term.

Weight	Age
16.08	1
33.83	2
65.80	3
97.20	4
191.55	5
326.20	6
386.87	7
520.53	8
590.03	9
651.92	10
724.93	11
699.56	12
689.96	13
637.56	14
717.41	15

Table 11.5: Body weights of rats at 15 ages. Source: Ratkowsky, D.A. (1983). Nonlinear Regression Modeling. New York, NY: Marcel Dekker, pp. 62 and 88.

The statements for fitting the logistic growth function are

```
D=Data();
D.input("rat.dat","y x");
Logistic=Model();
//          A          t*          b
Logistic.equation("y=intercept,x=intercept,x=intercept");
Logistic.param(-1); //m=-1
Init=Vector(700,7,.75);
Logistic.init(Init);
Logistic.link("richards",1)
Logistic.variance("residual",identity(3,3)*1000);
Logistic.fitdata(D);
Logistic.glim(20)
Logistic.vce_aireml(40)
K=identity(3,3);
Logistic.contrast(K)
Logistic.save("rat_log.out")
```

The `Logistic.param(-1);` statement is used to fix the shape parameter at  $-1$ . The model statement is used to describe the linear predictors for the three growth curve parameters. In addition, the first dependent variable is used to identify the weight variable and the second dependent variable is used to identify the age variable. The estimated growth curve parameters are  $A = 702.871 \pm 13.9397$ ,  $t^* = 6.4519 \pm 0.143226$ , and  $b = 0.688566 \pm 0.057372$ . The fitted curve is illustrated in Figure 11.3. The Richards model is describe

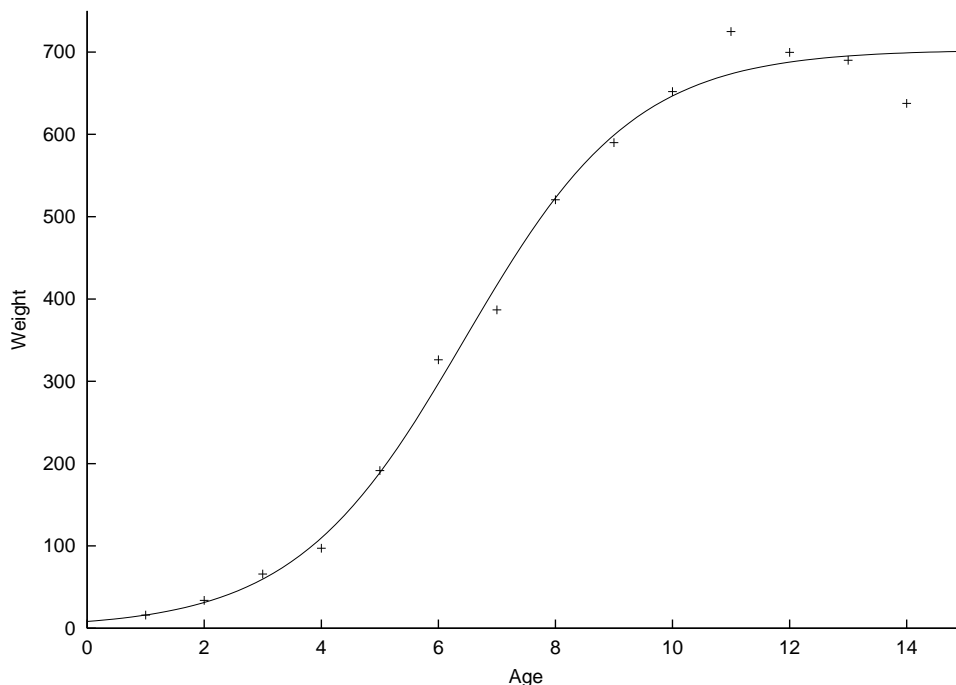


Figure 11.3: Logistic growth curve

similarly be adding a fourth linear predictor to the model equation and by dropping the `.param()` statement. The program is given below:

```
Richards=Model();
//          A          t*          b          m
```

Table 11.6: Number of positive tubes out of 10 in a replicated dose response data.

Log 2 Dose	Formulation A					Formulation B				
	Replication					Replication				
	1	2	3	4	5	1	2	3	4	5
0	2	9	9	7	8	9	8	9	10	9
1	2	6	8	8	10	7	7	8	8	9
2	2	6	6	8	7	0	1	3	7	7
3	1	3	1	6	6	0	2	0	1	1
4	1	6	0	3	4	0	1	0	0	1
5	0	7	2	1	4	0	0	0	0	0
6	1	1	1	3	3	0	0	0	0	0

```
Richards.equation("y=intercept,x=intercept,x=intercept,x=intercept");
Init=Vector(700,7,.75,-1);
Richards.init(Init);
Richards.link("richards",1)
Richards.variance("residual",identity(4,4)*1000);
Richards.fitdata(D);
Richards.glim(20)
Richards.vce_aireml(40)
K=identity(4,4);
Richards.contrast(K)
Richards.save("rat_ric.out")
```

## 11.2 Penalized Quasi-Likelihood

`M.vce_aireml()`<sup>1</sup> estimates variance components based on the penalized quasi-likelihood function. Consider an experiment similar to the dose response experiment except that it has been replicated 5 times. The data are in Table 11.6.

The linear predictor will include as fixed effects an intercept, a main effect for formulation, a covariate for log dose, and an interaction between formulation and log dose. Replication will be included as a random effect. Penalized quasi-likelihood estimates of the variance components are obtained using:

```
D=Data();
D.input("dose_r.dat","rep formul $ logdose n npos perc_pos");
M=Model();
M.equation("perc_pos=intercept formul logdose formul*logdose rep");
M.covariate("logdose");
M.weight("n");
M.variance("rep",2);
M.link("logit",0)
M.fitdata(D);
M.glim();
M.vce_aireml(25,0);
M.info("dose_r.info")
```

`M.vce_aireml()` takes up to two arguments. The first argument is the maximum number of iterations. The default number of iterations is 10. The second argument is the stopping criteria. By default the stopping criteria is to stop when residual log quasi-likelihood is changing by less than  $1.e - 4$  and will

<sup>1</sup>`M.vce_aireml()` is only available for models that include a link function. However, `M.link("normal",rvs)` can be used to specify a linear mixed model.

reduce the step size if the residual log quasi-likelihood decreases by more than  $1.e - 4$ . If the second argument is zero, then the stopping criteria is `EPSILON` (the default value of `EPSILON` is  $1.e - 14$ ) and it does not reduce the step size when the log likelihood increases. The `EPSILON` parameter can be set using `this.parameter("EPSILON", value)`. `M.vce_aireml()` returns a  $nr \times (nr + 1)$  matrix, where  $nr$  is the number of variance components. The first column of the matrix contains the estimated variance components. The remaining columns of the matrix contains the asymptotic covariance matrix.

The `M.info()` is similar to `M.save()` except it does not include the estimated fixed effects or predicted random effects. The resulting file `dose_r.info`:

```

                some extra information in the model
-----
AI REML converged
maximum log restricted likelihood = -297.196
variance for rep =
0.801036
residual variance =
1

MME dimension    : 11
non-zeros in MME: 52

                basic statistics for dependent variables
-----
      trait-name      n      mean      std
perc_pos              70      0.371429    0.348139
-----
```

The residual log quasi-likelihood can also be obtained using `M.residual.log_like()`.

### 11.2.1 Correlated random effects

In the previous example, it was assumed that the only effect of blocks was a change in the intercept. An alternative model would also allow the slope to change also. In the following Matvec code a random block by logdose is added to the model.

```

M=Model();
M.equation("perc_pos=intercept formul logdose formul*logdose rep"+\
          "rep*logdose");
M.covariate("logdose");
M.weight("n");
M.variance("rep",2);
M.variance("rep*logdose",.1);
M.link("logit",0)
M.fitdata(D);
```

However, the random intercept and random slope may be correlated. `M.variance(effect 1, effect 2, covariance)`<sup>2</sup> is used to designate that the random effects are correlated. The complete Matvec code is now:

```

D=Data();
D.input("dose_r.dat","rep formul $ logdose n npos perc_pos");
M=Model();
M.equation("perc_pos=intercept formul logdose formul*logdose rep"+\
          " rep*logdose");
M.covariate("logdose");
```

<sup>2</sup>Correlated random effects do not require a link function.



```

M.weight("n");
M.variance("rep",2);
M.variance("rep*logdose",.1);
M.variance("rep","rep*logdose",.01);
M.link("logit",0)
M.fitdata(D);
M.glim(10);
M.vce_airyml(25,0);
M.contrast("formul",[1,-1]);
M.info("dose_rr.info")

```

The resulting dose\_rr.info:

```

              some extra information in the model
-----
AI REML converged
maximum log restricted likelihood = -295.803
variance for rep =
0.892352 -0.0532517
-0.0532517 0.029527
residual variance =
1

MME dimension    : 16
non-zeros in MME: 92

      basic statistics for dependent variables
-----
      trait-name      n      mean      std
perc_pos              70      0.371429    0.348139
-----

```

### 11.2.2 Multivariate Correlated Random Effects

Data on body weight at 6 weeks of age and adjusted 4-6 week feed intake on 284 mice<sup>3</sup> will be used to illustrate a multi-trait analysis with correlated random effects. The model includes fixed effects for generation, litter size, and sex. The random effects include direct and maternal genetic effects and a litter effect. The structures of the covariance matrices are

$$\mathbf{G} = \begin{pmatrix} \sigma_{a1}^2 & \sigma_{a12} & \sigma_{a1m1} & \sigma_{a1m2} \\ \sigma_{a1a2} & \sigma_{a2}^2 & \sigma_{a2m1} & \sigma_{a2m2} \\ \sigma_{a1m1} & \sigma_{a2m1} & \sigma_{m1}^2 & \sigma_{m12} \\ \sigma_{a1m2} & \sigma_{a2m2} & \sigma_{m12} & \sigma_{m2}^2 \end{pmatrix}, \mathbf{C} = \begin{pmatrix} \sigma_{c1}^2 & \sigma_{c12} \\ \sigma_{c12} & \sigma_{c2}^2 \end{pmatrix}, \text{ and } \mathbf{R} = \begin{pmatrix} \sigma_{e1}^2 & \sigma_{e12} \\ \sigma_{e12} & \sigma_{e2}^2 \end{pmatrix}.$$

Using the following variance component estimates,

$$\mathbf{G} = \begin{pmatrix} 4.2 & -.08 & .41 & -.12 \\ -.08 & 5.9 & 1.4 & -.65 \\ .41 & 1.4 & 1.7 & -1.7 \\ -.12 & -.65 & -1.7 & 2.1 \end{pmatrix}, \mathbf{C} = \begin{pmatrix} .25 & .24 \\ .24 & 1.8 \end{pmatrix}, \text{ and } \mathbf{R} = \begin{pmatrix} 2.1 & 2.5 \\ 2.5 & 1.8 \end{pmatrix},$$

BLUEs and BLUPs can be obtained with the following code:

<sup>3</sup> Meyer, K. (1991). Estimating variances and covariances for multivariate Animal Models by Restricted Maximum Likelihood. *Genet. Sel. Evol.* **23**: 67-83.

```

D=Data();
D.input("mouse/mouse_2.dat",\
      " animal sire dam   byr sex littersize siregroup _skip wt_6wk"+\
      " fd_int directpe dampe");
P=Pedigree();
P.input("mouse/mouse.ped","individual father mother");
M=Model();
M.equation("wt_6wk=intercept byr sex littersize animal dam dampe,"+\
      "fd_int = intercept byr sex littersize animal dam dampe");
R=[2.1 2.5
   2.5 13];
GD=[4.2, -.08
    -.08 5.9];
GM=[1.7 , -1.7
    -1.7 2.1];
GDM=[.41 ,-.12
     1.4,-.65];
C=[.25 .24
   .24 1.8];
M.variance("residual",R)
M.variance("animal",P,GD)
M.variance("dam",P,GM)
M.variance("animal","dam",GDM)
M.variance("dampe",C)
M.fitdata(D);
M.blup();
M.save("mouse3a.out","out");

```

A portion of mouse3a.out:

```

               some extra information in the model
-----
variance for animal =
  4.20000 -0.0800000 0.410000 -0.120000
 -0.0800000  5.90000  1.40000 -0.650000
  0.410000  1.40000  1.70000 -1.70000
 -0.120000 -0.650000 -1.70000  2.10000
variance for dampe =
  0.250000 0.240000
  0.240000  1.80000
residual variance =
  2.10000  2.50000
  2.50000 13.0000

MME dimension   : 1466
non-zeros in MME: 22059
. . . . .
      Term      Trait 1      Trait 2
animal wt_6wk fd_int
      13c      0.528968      0.473886
      13r     -0.267458      0.581903
      16b     -0.267458      0.581903
. . . . .
      Term      Trait 1      Trait 2

```

```

dam wt_6wk fd_int
      13c      0.275847    -0.172521
      13r     -0.319031     0.506760
      16b     -0.319031     0.506760

```

REML estimates can be obtained with:

```

this.parameter("EPSILON",1.e-8);
M.link("normal",3);
est=M.vce_aireml(30,1.e-8);
vc=est(*,1);
cov=est(*,2:est.ncol());
se=cov.diag(0).sqrt();
"\nVariance Estimates and Standard Errors\n"
[vc se]
M.info("mouse3.info","out");
"Log Like",M.residual_log_like()

```

The `M.link("normal",3)` is needed by `M.vce_aireml()`, 3 is for the 3 residual variance components in a 2-trait residual covariance matrix. The `EPSILON` was set to larger value for numerical stability. The code:

```

vc=est(*,1);
cov=est(*,2:est.ncol());
se=cov.diag(0).sqrt();
"\nVariance Estimates and Standard Errors\n"
[vc se]

```

calculates asymptotic standard errors and prints the estimated variance components along with their standard errors. The abbreviated output:

Original Residual Log Likelihood:-2200.19

```

Iteration 1.0 Res Log Like -2200.08 Change 0.114878
Iteration 2.0 Res Log Like -2200.06 Change 0.0188999
Iteration 3.0 Res Log Like -2200.06 Change 0.000345039
Iteration 4.0 Res Log Like -2200.06 Change 3.24196e-05
Iteration 5.0 Res Log Like -2200.06 Change 4.6876e-06
Iteration 6.0 Res Log Like -2200.06 Change 8.18218e-07
Iteration 7.0 Res Log Like -2200.06 Change 1.57462e-07
Iteration 8.0 Res Log Like -2200.06 Change 3.17959e-08
Iteration 9.0 Res Log Like -2200.06 Change 6.70434e-09

```

Iteration 9 Converged

. . . . .

Variance Estimates and Standard Errors

```

4.05386 2.68904
-0.4364 2.80002
0.412177 1.81148
-0.0398331 2.12659
5.60885 6.16926
1.36832 2.29069
-0.432407 3.66772
1.58627 2.28487

```

```
-1.58156  1.89349
  1.8521  3.93396
```

The file mouse3.info:

```

               some extra information in the model
-----
AI REML converged
maximum log restricted likelihood = -2200.06
variance for animal =
  4.05386  -0.4364  0.412177 -0.0398331
 -0.4364   5.60885  1.36832 -0.432407
  0.412177  1.36832  1.58627 -1.58156
 -0.0398331 -0.432407 -1.58156   1.8521
variance for dampe =
  0.306701  0.163672
  0.163672  1.93397
residual variance =
   2.0784  2.74535
  2.74535  12.8342
```

### 11.2.3 Cubic splines

Data from a ELISA calibration study<sup>4</sup> and a constant variance with cubic splines model<sup>5</sup> will be used to illustrate the use of cubic splines and the `M.var_link()` function. The model includes a fixed covariate for concentration, correlated random effects for trial and trial\*concentration, and random effects for splines and splines by trial. The model equation is

$$od_{ijk} = \mu + \beta conc_{ij} + \sum_{\ell=1}^5 \gamma_{\ell} u_{ij\ell} + T_i + \beta_i conc_{ij} + \sum_{\ell=1}^5 \gamma_{i\ell} u_{ij\ell} + e_{ijk}$$

where  $od_{ijk}$  is the optical density for trial  $i$  with concentration  $conc_j$  and sub-sample  $k$ ,  $\mu$  is the intercept,  $\beta$  is the overall slope,  $\beta_i \sim N(0, \sigma_C^2)$  is the random trial specific slope,  $u_{ij\ell}$  is the  $\ell$  spline covariate,  $\gamma_{\ell} \sim N(0, \sigma_s^2)$  is the overall  $\ell$  spline random effect,  $\gamma_{i\ell} \sim N(0, \sigma_{st}^2)$  is the  $\ell$  spline random effect for trial  $i$ ,  $T_i \sim N(0, \sigma_T^2)$  is the trial random effect and is correlated with  $\beta_i$  with covariance  $\sigma_{CT}$ , and  $e_{ijk} \sim N(0, \sigma^2)$  is the residual.

To fit a model with cubic splines the first step is to compute the spline covariates and this is accomplished with the following code

```
D=Data();
D.input("../data/elisa.dat","trial conc od");
knots=[0 pow(2,-6:0)*12.5].t()
nk=knots.nrow();
Dmat=D.mat();
conc=Dmat(:,3);
X=conc.splines(knots,1);
X.save("elisa-splines.dat");
```

Next the covariate data is merged with the original data

```
system("paste -d\" \" ../data/elisa.dat elisa-splines.dat > elisa-fit.dat");
```

The model is then specified

<sup>4</sup> Davidian, M. and D. M. Giltinan (1993). Some simple methods for estimating intraindividual variability in nonlinear mixed effects models. *Biometrics* **49**: 59-73.

<sup>5</sup> Verbyla, A. P., B. R. Cullis, M. G. Kenward and S. J. Welham (1999). The analysis of designed experiments and longitudinal data by using smoothing splines. *Appl.Statist.* **48**: 269-311

```

u=" ";
tu=" ";
for(i=1;i<=(nk-2);i++){
    u=string(u," u",i);
    tu=string(tu," trial*u",i);
}
D.input("elisa-fit.dat",string("trial conc od int x ",u));
M=Model();
M.equation(string("od=intercept trial conc ",u," trial*conc ",tu));
M.covariate("conc");
M.variance("trial",5);
M.variance("trial*conc",1);
M.variance("trial","trial*conc",.01);
for(i=1;i<=(nk-2);i++){
    M.covariate(string("u",i));
    M.variance(string("u",i),100);
    M.variance(string("trial*u",i),1);
}
M.variance("residual",4);

```

Next the two sets of five spline random effects share a common variance they are linked together

```

for(i=2;i<=(nk-2);i++){
    M.var_link("u1",string("u",i));
    M.var_link("trial*u1",string("trial*u",i));
}

```

Finally the model is completed and the analysis is run

```

M.link("normal",1);
M.fitdata(D);
M.glim();
M.vce_aireml(200,0);
M.save("elisa.out");

```



## Chapter 12

# Segregation and Linkage Analyses

### 12.1 Genotype Probability Computation

It is very difficult, in general, to compute the genotype probabilities for each member of a pedigree with loops . Van Arendonk et al. (1889), Janss et al. (1992), Fernando et al. (1993), and Wang et al. (1995) discussed and proposed an iterative algorithm to compute the genotype probabilities for large and complex livestock pedigrees. The proposed iterative algorithms are based on primary works by Murphy and Mutalik (1969), Elston and Stewart (1971), and Heuch and Li (1972).

The conditional probability that pedigree member  $i$  has genotype  $u_i$  given all the phenotypes  $y_1, \dots, y_n$  can be computed as

$$\Pr(u_i | y_1, \dots, y_n) = \frac{a_i(u_i)g(y_i | u_i) \prod_{j \in S_i} \{p_{ij}(u_i)\}}{\sum_{u_i} a_i(u_i)g(y_i | u_i) \prod_{j \in S_i} \{p_{ij}(u_i)\}} \quad (12.1)$$

where  $a_i(u_i)$  is the joint probability of phenotypes of members anterior to  $i$  and of genotype  $u_i$  for  $i$ ,  $g(y_i | u_i)$  is the penetrance function, and  $\prod_{j \in S_i} \{p_{ij}(u_i)\}$  is the conditional probability of phenotypes of members posterior to  $i$ , given  $i$  has genotype  $u_i$ .

The anterior probability,  $a_i(u_i)$ , and posterior probability,  $p_{ij}(u_i)$ , can be computed as

$$\begin{aligned} a_i(u_i) &= \sum_{u_m} \{a_m(u_m)g(y_m | u_m) \prod_{j \in S_m, j \neq f} p_{mj}(u_m) \\ &\times \sum_{u_f} \{a_f(u_f)g(y_f | u_f) \prod_{j \in S_f, j \neq m} p_{fj}(u_f) \\ &\times tr(u_i | u_m, u_f) \\ &\times \prod_{j \in C_{mf}, j \neq i} \left[ \sum_{u_j} tr(u_j | u_m, u_f)g(y_j | u_j) \prod_{k \in S_j} p_{jk}(u_j) \right] \} \end{aligned} \quad (12.2)$$

and

$$\begin{aligned} p_{ij}(u_i) &= \sum_{u_j} \{a_j(u_j)g(y_j | u_j) \prod_{k \in S_j, k \neq i} p_{jk}(u_j) \\ &\times \prod_{k \in C_{ij}} \left[ \sum_{u_k} tr(u_k | u_i, u_j)g(y_k | u_k) \prod_{l \in S_k} p_{kl}(u_k) \right] \} \end{aligned} \quad (12.3)$$

where  $C_{ij}(C_{mf})$  is the set of offspring of parents  $i$  and  $j$  ( $m$  and  $f$ ).

The step-by-step iterative algorithm is given below:

1. For each member  $i$ :

- (a) Initialize anterior values: set anterior values equal to the population genotype frequencies and set the anterior log scaling factor equal to zero,
  - (b) Initialize posterior values: set posterior values equal to unity and set the posterior log scaling factor equal to zero, and
  - (c) Compute penetrance values and the corresponding log scaling factor.
2. For each connector  $i$ :
  - (a) For families in which  $i$  is an offspring, compute its anterior value  $a_i(u_i)$  using current values of required quantities.
  - (b) For families in which  $i$  is a parent, compute its posterior value through each mate  $j$ ,  $p_{ij}$ , using current values of the required quantities.
3. Repeat step 2 (usually less than 10 times) until each of the anterior and posterior values for each member has converged.

## 12.2 Genetic Mapping

not completed yet.



# Chapter 13

## Statistical Distributions

### 13.1 Introduction

An object of StatDist class can be created as below:

```
D = StatDist("name",parameter-list)
```

where

1. **name** is one of the followings: Uniform, Normal, ChiSquare, t, F, Gamma, Exponential, Beta, DiscreteUniform, Poisson, Binomial, NegativeBinomial, Geometric. Hypergeometric, Logistic, Laplace, InverseGaussian, Cauchy, Weibull, Erlang will be added into the list.
2. **parameter-list** is a list of scalar values required by the corresponding distribution.

Note that some distributions have their non-centrality parameters. The non-centrality parameter is always the last one in the parameter list. It is automatically set to zero if users don't provide it explicitly.

There are nine member functions available now.

pdf D.pdf(x) returns the probability density function values of x which could be a vector or matrix.

cdf D.cdf(x) returns the cumulative distribution function values of x which could be a vector or matrix

mgf D.mgf(t) returns the moment-generating function values of t which could be a vector or matrix.

inv  $x = D.inv(p)$  is the inverse function of  $D.cdf(x)$ , where p could be a vector or matrix.

nonct D.nonct(cv,p) returns non-centrality value given the critical value (cv) and p value (cdf). Both cv and p could be either vector or matrix as long as the sizes are the same.

sample D.sample(), D.sample(n), and D.sample(m,n) return a random scalar or a vector of size n or a matrix of size m by n.

parameter D.parameter(i) returns the i'th parameter of the distribution.

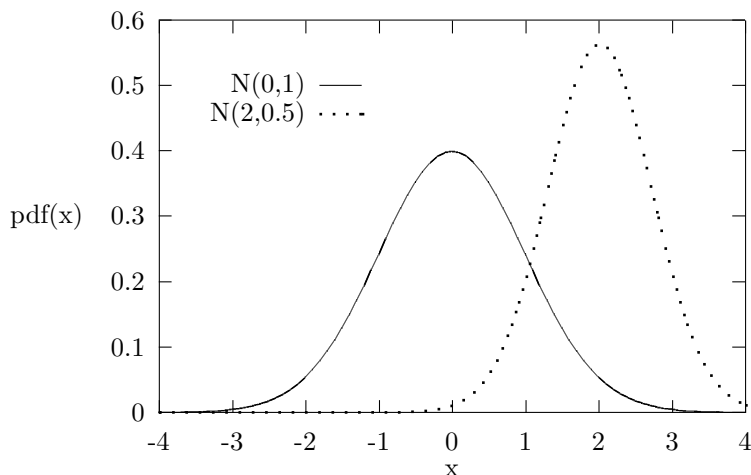
mean D.mean() returns the first moment (expected value) of the distribution.

variance D.variance() returns the second central moment (variance) of the distribution.

Here are two examples:

```
> N = StatDist("Normal",0, 10);  
> B = StatDist("Binomial",2, 0.1);
```

Figure 13.1: Normal Probability Density Function (pdf)



## 13.2 Continuous Distribution

The distributions for continuous random variables are Uniform, Normal, ChiSquare, t, F, Gamma, Beta, Exponential, and LogNormal.

### 13.2.1 Normal distribution

#### Definition

The random variable  $X$  has a *normal distribution* if its probability density function (pdf) is defined by

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad -\infty < x < \infty \quad (13.1)$$

where  $\mu$  (real) and  $\sigma^2$  (real) are the parameters with their range  $-\infty < \mu < \infty, \sigma^2 > 0$ . In short, we say  $X \sim N(\mu, \sigma^2)$ . The normal distribution with  $\mu = 0$  and  $\sigma^2 = 1$  is known as the standard normal distribution  $N(0, 1)$ .

The graph of normal pdf function for two sets of parameters is shown in Fig. 13.1. The Matvec script used to draw this graph is on page 50.

The normal distribution is certainly the most important distribution in statistical data analyses since many performance measurements such as milkyield have (approximate) normal distributions.

#### Properties

1. The normal distribution is scalable. If  $Z \sim N(0, 1)$  then  $\sigma Z + \mu \sim N(\mu, \sigma^2)$ , and conversely if  $X \sim N(\mu, \sigma^2)$ , then  $\frac{X-\mu}{\sigma} \sim N(0, 1)$ .
2. the distribution is symmetric
3. its moment generation function (mgf) is

$$M(t) = \exp(\mu t + \sigma^2 t^2 / 2), \quad -\infty < t < \infty$$

4. the cumulative distribution function (cdf) for  $X \sim N(0, 1)$  is

$$\begin{aligned}\Phi(x) &= \int_{-\infty}^x \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{w^2}{2}\right) dw \\ &= \begin{cases} \frac{1+\operatorname{erf}(x/\sqrt{2})}{2} & x \geq 0 \\ \frac{1-\operatorname{erf}(-x/\sqrt{2})}{2} & x < 0 \end{cases}\end{aligned}$$

5.  $E(X) = \mu$ ,  $\operatorname{Var}(X) = \sigma^2$ .

6. the median  $= \mu$ .

7. if  $X_1, X_2, \dots, X_n$  is a sample from  $N(\mu, \sigma^2)$ , let

$$\bar{X} = \frac{\sum_{i=1}^n X_i}{n}, \text{ and } s^2 = \frac{\sum_{i=1}^n (X_i - \bar{X})^2}{n-1}$$

then  $\bar{X} \sim N(\mu, \frac{\sigma^2}{n})$ ,  $(n-1)\frac{s^2}{\sigma^2} \sim \chi^2(n-1)$ , and both are independent.

### Matvec interface

An object of  $N(\mu, \sigma^2)$  can be created by

```
D = StatDist("Normal",mu,sigma2);
D = StatDist("Normal");           // standard normal N(0,1)
```

Matvec provided several standard member functions to allow user to access most of properties and functions of  $N(\mu, \sigma^2)$ :

pdf  $D.\text{pdf}(x)$  returns the probability density function (pdf) values of  $x$  which could be a vector or matrix.

cdf  $D.\text{cdf}(x)$  returns the cumulative distribution function (cdf) values of  $x$  which could be a vector or matrix

mgf  $D.\text{mgf}(t)$  returns the moment-generating function (mgf) values of  $t$  which could be a vector or matrix.

inv  $D.\text{inv}(p)$  is the inverse function of  $D.\text{cdf}(x)$ , where  $p$  could be a vector or matrix. That is if  $p = D.\text{cdf}(x)$ , then  $x = D.\text{inv}(p)$ .

sample  $D.\text{sample}()$ ,  $D.\text{sample}(n)$ , and  $D.\text{sample}(m,n)$  return a random scalar or a vector of size  $n$  or a matrix of size  $m$  by  $n$ .

parameter  $D.\text{parameter}(1)$  returns  $\mu$  and  $D.\text{parameter}(2)$  returns  $\sigma^2$ .

mean  $D.\text{mean}()$  returns the expected value of  $X \sim N(\mu, \sigma^2)$ .

variance  $D.\text{variance}()$  returns the variance of  $X \sim N(\mu, \sigma^2)$ .

### Examples

```
> D = StatDist("Normal",2, 10)
      NormalDist(2,10)
> D.mean()
      2
> D.pdf([-2,0,2])
      Col 1      Col 2      Col 3
Row 1    0.0566858  0.103288  0.126157
> D.cdf([-3.20148,2,7.20148])
```

```

          Col 1      Col 2      Col 3
    Row 1    0.0500001    0.500000    0.950000
> D.inv([0.05,0.5,0.95])
          Col 1      Col 2      Col 3
    Row 1   -3.20148     2.00000     7.20148
> D.sample(2,3)
          Col 1      Col 2      Col 3
    Row 1   -1.26660     2.85846     5.64868
    Row 2     3.51262    -1.97644     3.44860

```

### 13.2.2 Uniform distribution

#### Definition

The random variable  $X$  has a *uniform distribution* if its probability density function (pdf) is defined by

$$f(x) = \frac{1}{b-a}, \quad a \leq x \leq b \quad (13.2)$$

where  $a$  (real) and  $b$  (real) are the parameters with their range  $-\infty < a < b < \infty$ . In short, we say  $X \sim U(a, b)$ . The uniform distribution with  $a = 0$  and  $b = 1$  is known as the standard uniform distribution  $U(0, 1)$ . One of interesting example is that if  $D$  is an object of `StatDist`, then  $D.cdf(x)$  is distributed as  $U(0, 1)$ .

#### Properties

1.  $E(X) = \frac{a+b}{2}$ ,  $Var(X) = \frac{(b-a)^2}{12}$ .
2. its moment generation function is

$$M(t) = \frac{e^{tb} - e^{ta}}{t(b-a)}, \quad t \neq 0; M(0) = 1$$

#### Matvec interface

An object of  $U(a, b)$  can be created by

```

D = StatDist("Uniform",a,b);
D = StatDist("Uniform");           // standard uniform U(0,1)

```

Matvec provided several standard member functions to allow user to access most of properties and functions of  $U(a, b)$ :

pdf  $D.pdf(x)$  returns the probability density function (pdf) values of  $x$  which could be a vector or matrix.

cdf  $D.cdf(x)$  returns the cumulative distribution function (cdf) values of  $x$  which could be a vector or matrix

mgf  $D.mgf(t)$  returns the moment-generating function (mgf) values of  $t$  which could be a vector or matrix.

inv  $D.inv(p)$  is the inverse function of  $D.cdf(x)$ , where  $p$  could be a vector or matrix. That is if  $p = D.cdf(x)$ , then  $x = D.inv(p)$ .

sample  $D.sample()$ ,  $D.sample(n)$ , and  $D.sample(m,n)$  return a random scalar or a vector of size  $n$  or a matrix of size  $m$  by  $n$ .

parameter  $D.parameter(1)$  returns  $a$  and  $D.parameter(2)$  returns  $b$ .

mean  $D.mean()$  returns the expected value of  $X \sim U(a, b)$ .

variance  $D.variance()$  returns the variance of  $X \sim U(a, b)$ .

**Examples**

```

> D = StatDist("Uniform",1, 4)
      UniformDist(1,4)
> D.variance()
      0.75
> D.pdf([1,2,3,4])
      Col 1      Col 2      Col 3      Col 4
Row 1      0.333333      0.333333      0.333333      0.333333
> D.cdf([1,2,3,4])
      Col 1      Col 2      Col 3      Col 4
Row 1      0.000000      0.333333      0.666667      1.000000
> D.inv([0,1/3,2/3,1])
      Col 1      Col 2      Col 3      Col 4
Row 1      1.000000      2.000000      3.000000      4.000000
> U.sample(1,3)
      Col 1      Col 2      Col 3
Row 1      3.59627      1.36787      2.17816

```

**13.2.3  $\chi^2$  distribution****Definition**

The random variable  $X$  has a non-central  $\chi^2$  distribution if its probability density function (pdf) is defined by

$$f(x) = \frac{\exp(-(x + \lambda)/2)}{2^{\frac{1}{2}}r} \sum_{j=0}^{\infty} \frac{x^{r/2+j-1} \lambda^j}{\Gamma(r/2 + j) 2^{2j} j!}, \quad 0 \leq x < \infty \quad (13.3)$$

where  $r$  (integer) and  $\lambda$  (real) are the parameters with their ranges  $r \geq 1, \lambda \geq 0$ . The parameter  $r$  is commonly called the degrees of freedom and  $\lambda$  non-centrality parameter. In short, we say  $X \sim \chi^2(r, \lambda)$ .

Alternatively, If  $Z_1, Z_2, \dots, Z_r$  are independent and distributed as  $N(\delta_i, 1)$  then the random variable

$$X = \sum_{i=1}^r Z_i^2$$

is called the non-central  $\chi^2$  distribution with  $r$  degrees of freedom and non-centrality parameter  $\lambda = \sum_{i=1}^r \delta_i$ .

When  $\lambda = 0$ , the p.d.f of  $\chi^2(r, \lambda)$  reduces to

$$f(x) = \frac{1}{\Gamma(r/2) 2^{r/2}} x^{r/2-1} e^{-x/2}, \quad (13.4)$$

we say  $X \sim \chi^2(r)$  which is commonly called (central)  $\chi^2$  distribution.

The central  $\chi^2$  distribution is the special case of Gamma distribution:  $\chi^2(r) = \Gamma(r/2, 2)$

For example, if  $X_1, X_2, \dots, X_n$  is a sample from  $N(\mu, \sigma^2)$ , then

$$(n-1) \frac{s^2}{\sigma^2} \sim \chi^2(n-1)$$

and is independent of the sample mean  $\bar{X}$ .

**Properties**

1. its moment generation function is

$$M(t) = (1 - 2t)^{-r/2} \exp\left(\frac{\lambda t}{1 - 2t}\right), \quad t < \frac{1}{2}$$

2.  $E(X) = r + \lambda$ ,  $\text{Var}(X) = 2(r + 2\lambda)$ .

3. If  $\chi^2(r_1, \lambda_1)$  and  $\chi^2(r_2, \lambda_2)$  are independent, then  $\chi^2(r_1, \lambda_1) + \chi^2(r_2, \lambda_2) = \chi^2(r_1 + r_2, \lambda_1 + \lambda_2)$

**Matvec interface**

An object of  $\chi^2(r, \lambda)$  can be created by

```
D = StatDist("ChiSquare",r,lambda);
D = StatDist("ChiSquare",r);
```

Matvec provided several standard member functions to allow user to access most of properties and functions of  $\chi^2(r, \lambda)$ :

pdf D.pdf(x) returns the probability density function (pdf) values of x which could be a vector or matrix.

cdf D.cdf(x) returns the cumulative distribution function (cdf) values of x which could be a vector or matrix

mgf D.mgf(t) returns the moment-generating function (mgf) values of t which could be a vector or matrix.

inv D.inv(p) is the inverse function of D.cdf(x), where p could be a vector or matrix. That is if  $p = D.cdf(x)$ , then  $x = D.inv(p)$ .

nonct D.nonct(cv,p) returns non-centrality value given the critical value (cv) and p value (cdf). Both cv and p could be either vector or matrix as long as the sizes are the same.

sample D.sample(), D.sample(n), and D.sample(m,n) return a random scalar or a vector of size n or a matrix of size m by n.

parameter D.parameter(1) returns  $r$ , degree of freedom.

mean D.mean() returns the expected value of  $X \sim \chi^2(r, \lambda)$ .

variance D.variance() returns the variance of  $X \sim \chi^2(r, \lambda)$ .

**Examples**

```
> D = StatDist("ChiSquare",10,0.1)
      ChiSquareDist(10,0.1)
> D.mean()
      10.1
> D..sample(1000).mean()
      10.0323
> D.pdf(5)

***ERROR***
ChiSquareDist::pdf(): not available yet: non-centrality
> D.cdf([0,1,10])
      Col 1      Col 2      Col 3
Row 1      0.00000  0.000164396  0.550770
> D.inv([0,0.000164396,0.550770])
      Col 1      Col 2      Col 3
Row 1      0.00000      1.00000     10.0000
> D.nonct(10,0.550770)
      0.100003
> D.sample()
      5.15093
```

### 13.2.4 $t$ distribution

#### Definition

The random variable  $X$  has a *non-central  $t$  distribution* if its probability density function (pdf) is defined by

$$f(x) = \frac{\exp(\delta^2/2)\Gamma((r+1)/2)}{\sqrt{\pi r}\Gamma(r/2)} \left(\frac{r}{r+x^2}\right)^{(r+1)/2} \quad (13.5)$$

$$\times \sum_{j=0}^{\infty} \frac{\Gamma((r+j+1)/2)}{j!\Gamma((r+1)/2)} \left[ \frac{x\delta\sqrt{2}}{\sqrt{r+x^2}} \right]^j, \quad -\infty < x < \infty \quad (13.6)$$

where  $r$  (integer) and  $\delta$  (real) are the parameters with their ranges  $r \geq 1, -\infty < \delta < \infty$ ;  $r$  is commonly called the degrees of freedom and  $\delta$  non-centrality parameter. In short, we say  $X \sim t(r, \delta)$ .

Alternatively, If  $Z \sim N(\delta, 1)$  and  $U \sim \chi^2(r)$  are independent then the random variable

$$X = \frac{Z}{\sqrt{U/r}}$$

is called the non-central  $t$  distribution with  $r$  degrees of freedom and non-centrality parameter  $\delta$ .

When  $\delta = 0$ , the p.d.f of  $t(r, \delta)$  reduces to

$$f(x) = \frac{\Gamma((r+1)/2)}{\sqrt{\pi r}\Gamma(r/2)} \left(\frac{r}{r+x^2}\right)^{(r+1)/2} \quad (13.7)$$

we say  $X \sim t(r)$  which is commonly called (central)  $t$  distribution.

For example, if  $X_1, X_2, \dots, X_n$  is a random sample from  $N(\mu, \sigma^2)$ , then

$$\frac{\bar{X} - \mu}{\sigma/\sqrt{n}} \sim N(0, 1), (n-1)\frac{s^2}{\sigma^2} \sim \chi^2(n-1),$$

and both are independent. Thus,

$$\frac{\bar{X} - \mu}{s/\sqrt{n}} \sim t(n-1)$$

Because

$$\frac{\bar{X}}{\sigma/\sqrt{n}} \sim N(\mu, 1)$$

Thus,

$$\frac{\bar{X}}{s/\sqrt{n}} \sim t(n-1, \frac{\mu}{\sigma/\sqrt{n}})$$

#### Properties

$$1. E(X) = (r/2)^{1/2} \frac{\Gamma((r-1)/2)}{\Gamma(r/2)} \delta, \text{ Var}(X) = \frac{r}{r-2} (1 + \delta^2) - E^2(X).$$

#### Matvec interface

An object of  $t(r, \delta)$  can be created by

```
> D = StatDist("t",r,delta)
> D = StatDist("t",r)
```

Matvec provided several standard member functions to allow user to access most of properties and functions of  $t(r, \delta)$ :

pdf D.pdf(x) returns the probability density function (pdf) values of  $x$  which could be a vector or matrix.

`cdf D.cdf(x)` returns the cumulative distribution function (cdf) values of  $x$  which could be a vector or matrix

`mgf D.mgf(t)` returns the moment-generating function (mgf) values of  $t$  which could be a vector or matrix.

`inv D.inv(p)` is the inverse function of `D.cdf(x)`, where  $p$  could be a vector or matrix. That is if  $p = D.cdf(x)$ , then  $x = D.inv(p)$ .

`nonct D.nonct(cv,p)` returns non-centrality value given the critical value ( $cv$ ) and  $p$  value (cdf). Both  $cv$  and  $p$  could be either vector or matrix as long as the sizes are the same.

`sample D.sample()`, `D.sample(n)`, and `D.sample(m,n)` return a random scalar or a vector of size  $n$  or a matrix of size  $m$  by  $n$ .

`parameter D.parameter(1)` returns  $r$ , degree of freedom; whereas `D.parameter(2)` returns  $\delta$ , non-centrality parameter.

`mean D.mean()` returns the expected value of  $X \sim t(r, \delta)$ .

`variance D.variance()` returns the variance of  $X \sim t(r, \delta)$ .

### Examples

```
> D = StatDist("t",15,2)
      tDist(15,2)
> D.mean()
      2.10746
> D.variance()
      1.32782
> D.pdf(2)

***ERROR***
tDist::pdf(): non-centrality: not available
> D.cdf([-1,0,1,5])
      Col 1      Col 2      Col 3      Col 4
Row 1  0.00165624  0.0227501  0.158591  0.986127
> D.inv([0.00165624,0.0227501, 0.158591,0.986127])
      Col 1      Col 2      Col 3      Col 4
Row 1   -1.00000 -0.000122502  0.999999  5.00002
> D.nonct(5,0.986127)
      1.99998
> D.sample(3)
      i = 1      i = 2      i = 3
0.952105      2.99387      0.707365
```

### 13.2.5 $F$ distribution

#### Definition

If  $U_1 \sim \chi^2(r_1, \delta)$  and  $U_2 \sim \chi^2(r_2)$  are independent then the random variable

$$X = \frac{U_1/r_1}{U_2/r_2}$$

is called the non-central  $F$  distribution with  $r_1$  (integer) and  $r_2$  (integer) degrees of freedom and non-centrality parameter  $\delta$  (real), denoted by  $F(r_1, r_2, \delta)$  where  $r_1, r_2 \geq 1$  and  $\delta \geq 0$ .



**Properties**

1.  $E(X) = \frac{r_2(r_1+\delta)}{r_1(r_2-2)}$ , for  $r_2 > 2$
2.  $\text{Var}(X) = 2(r_2/r_1)^2 \frac{(r_1+\delta)^2 + (r_1+2\delta)(r_2-2)}{(r_2-2)^2(r_2-4)}$  for  $r_2 > 4$

**Matvec interface**

An object of  $F(r_1, r_2, \delta)$  can be created by

```
D = StatDist("F",r1,r2,delta);
D = StatDist("F",r1,r2);
```

Matvec provided several standard member functions to allow user to access most of properties and functions of  $F(r_1, r_2, \delta)$ :

pdf D.pdf(x) returns the probability density function (pdf) values of x which could be a vector or matrix.

cdf D.cdf(x) returns the cumulative distribution function (cdf) values of x which could be a vector or matrix

mgf D.mgf(t) returns the moment-generating function (mgf) values of t which could be a vector or matrix.

inv D.inv(p) is the inverse function of D.cdf(x), where p could be a vector or matrix. That is if  $p = D.cdf(x)$ , then  $x = D.inv(p)$ .

nonct D.nonct(cv,p) returns non-centrality value given the critical value (cv) and p value (cdf). Both cv and p could be either vector or matrix as long as the sizes are the same.

sample D.sample(), D.sample(n), and D.sample(m,n) return a random scalar or a vector of size n or a matrix of size m by n.

parameter D.parameter(1) returns  $r_1$ , D.parameter(2) returns  $r_2$ , and D.parameter(3) returns  $\delta$ .

mean D.mean() returns the expected value.

variance D.variance() returns the variance.

**Examples**

```
> D = StatDist("F",5,100,0.5)
      FDist(5,100,0.5)
> D.mean()
      1.12245
> D.sample(1000).mean()
      1.11399
> D.variance()
      0.536452
> D.pdf(2)

***ERROR***
FDist::pdf(): not available yet: non-centrality
> D.cdf([0.1,1,3])
      Col 1      Col 2      Col 3
Row 1  0.00642900  0.520762  0.976739
> D.inv([0.00642900,0.520762,0.976739])
      Col 1      Col 2      Col 3
Row 1  0.100000  1.00000  3.00000
```

```
> D.mgf(2)

***ERROR***
FDist:mgf(): not available yet
> D.nonct(3,0.95)
1.626
```

### 13.2.6 Gamma distribution

#### Definition

The random variable  $X$  has a *gamma distribution* if its probability density function is defined by

$$f(x) = \frac{1}{\Gamma(\alpha)\theta^\alpha} x^{\alpha-1} e^{-x/\theta}, \quad 0 \leq x < \infty. \quad (13.8)$$

where  $\alpha$  (real) and  $\theta$  (real) are the parameters with their ranges  $\alpha, \theta > 0$ . In short, we say  $X \sim \text{Gamma}(\alpha, \theta)$ .

#### Properties

1. moment generating function

$$M(t) = \frac{1}{(1 - \theta t)^\alpha}, \quad t < 1/\theta$$

2.  $E(X) = \alpha\theta, \quad \text{Var}(X) = \alpha\theta^2$

#### Matvec interface

An object of  $\text{Gamma}(\alpha, \theta)$  can be created by

```
D = StatDist("Gamma",alpha,theta);
```

Matvec provided several standard member functions to allow user to access most of properties and functions of  $\text{Gamma}(\alpha, \theta)$ :

pdf D.pdf(x) returns the probability density function (pdf) values of x which could be a vector or matrix.

cdf D.cdf(x) returns the cumulative distribution function (cdf) values of x which could be a vector or matrix

mgf D.mgf(t) returns the moment-generating function (mgf) values of t which could be a vector or matrix.

inv D.inv(p) is the inverse function of D.cdf(x), where p could be a vector or matrix. That is if  $p = \text{D.cdf}(x)$ , then  $x = \text{D.inv}(p)$ .

sample D.sample(), D.sample(n), and D.sample(m,n) return a random scalar or a vector of size n or a matrix of size m by n.

parameter D.parameter(1) returns  $\alpha$ , D.parameter(2) returns  $\theta$ .

mean D.mean() returns the expected value.

variance D.variance() returns the variance.

**Examples**

```

> D = StatDist("Gamma",2,20)
      GammaDist(2,20)
> D.mean()
      40
> D.sample(1000).mean()
      40.3099
> D.pdf([1,10,100])
      Col 1      Col 2      Col 3
Row 1  0.00237807  0.0151633  0.00168449
> D.cdf([1,10,100])
      Col 1      Col 2      Col 3
Row 1  0.00120910  0.0902040  0.959572
> D.inv([0.00120910,0.0902040,0.959572])
      Col 1      Col 2      Col 3
Row 1  0.999998  10.0000  99.9998

```

**13.2.7 Exponential distribution****Definition**

The random variable  $X$  has a *exponential distribution* if its probability density function is defined by

$$f(x) = \frac{1}{\theta} e^{-x/\theta}, \quad 0 \leq x < \infty. \quad (13.9)$$

where  $\theta$  (real) is the parameter with its range  $\theta > 0$ . In short, we say  $X \sim E(\theta)$ . The exponential distribution with  $\theta = 1$  is known as the standard exponential distribution.

**Properties**

1. moment generating function

$$M(t) = \frac{1}{1 - \theta t}, \quad t < \frac{1}{\theta}$$

2.  $E(X) = \theta$ ,  $\text{Var}(X) = \theta^2$

3. the cumulative distribution function (cdf) is defined

$$F(x) = \begin{cases} 0 & -\infty < x < 0 \\ 1 - e^{-x/\theta} & 0 \leq x < \infty \end{cases}$$

4. the median  $m$  is found by solving  $F(m) = 0.5$ . That is  $m = \theta \log(2)$ . because  $\log(2) < 1$ . Thus the median is always less than the mean ( $\theta$ ).

**Matvec interface**

An object of  $E(\theta)$  can be created by

```

D = StatDist("Exponential",theta);
D = StatDist("Exponential");

```

Matvec provided several standard member functions to allow user to access most of properties and functions of  $E(\theta)$ :

pdf D.pdf(x) returns the probability density function (pdf) values of x which could be a vector or matrix.

cdf `D.cdf(x)` returns the cumulative distribution function (cdf) values of  $x$  which could be a vector or matrix.

mgf `D.mgf(t)` returns the moment-generating function (mgf) values of  $t$  which could be a vector or matrix.

inv `D.inv(p)` is the inverse function of `D.cdf(x)`, where  $p$  could be a vector or matrix. That is if  $p = D.cdf(x)$ , then  $x = D.inv(p)$ .

sample `D.sample()`, `D.sample(n)`, and `D.sample(m,n)` return a random scalar or a vector of size  $n$  or a matrix of size  $m$  by  $n$ .

parameter `D.parameter(1)` returns  $\theta$ .

mean `D.mean()` returns the expected value.

variance `D.variance()` returns the variance.

### Examples

```
> D = StatDist("Exponential",10)
      ExponentialDist(10)
> D.sample(1000).mean()
      9.69597
> D.pdf([0,2,10])
      Col 1      Col 2      Col 3
Row 1      0.100000      0.0818731      0.0367879
> D.cdf([0,2,10])
      Col 1      Col 2      Col 3
Row 1      0.00000      0.181269      0.632121
> D.inv([0,0.181269,0.632121])
      Col 1      Col 2      Col 3
Row 1      5.63450e-07      2.00000      10.0000
```

### 13.2.8 Beta distribution

#### Definition

The random variable  $X$  has a *non-central beta distribution* if its probability density function is defined by

$$f(x) = \dots \quad (13.10)$$

where  $\alpha$  (real),  $\beta$  (real) and  $\lambda$  (real) are the parameters with their ranges  $\alpha, \beta > 0$  and  $\lambda \geq 0$ . In short, we say  $X \sim \text{Beta}(\alpha, \beta, \lambda)$ .

If  $\lambda = 0$ , then its pdf reduces to

$$f(x) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1-x)^{\beta-1}, \quad 0 < x < 1 \quad (13.11)$$

we say  $X \sim \text{Beta}(\alpha, \beta)$  which is commonly called (central) *Beta* distribution.

Alternatively, if  $X_1 \sim \chi^2(r_1, \lambda)$  and  $X_2 \sim \chi^2(r_2)$  are independent, then the ratio

$$X = \frac{X_1}{X_1 + X_2} \quad (13.12)$$

is defined as a non-central beta distribution  $\text{Beta}(r_1/2, r_2/2, \lambda)$

#### Properties

1. for  $X \sim \text{Beta}(\alpha, \beta)$ ,  $E(X) = \frac{\alpha}{\alpha + \beta}$ ,  $\text{Var}(X) = \frac{\alpha\beta}{(\alpha + \beta + 1)(\alpha + \beta)^2}$ .

**Matvec interface**

An object of  $Beta(\alpha, \beta, \lambda)$  can be created by

```
D = StatDist("Beta",alpha,beta,lambda);
D = StatDist("Beta",alpha,beta);
```

Matvec provided several standard member functions to allow user to access most of properties and functions of  $Beta(\alpha, \beta, \lambda)$ :

pdf D.pdf(x) returns the probability density function (pdf) values of x which could be a vector or matrix.

cdf D.cdf(x) returns the cumulative distribution function (cdf) values of x which could be a vector or matrix

mgf D.mgf(t) returns the moment-generating function (mgf) values of t which could be a vector or matrix.

inv D.inv(p) is the inverse function of D.cdf(x), where p could be a vector or matrix. That is if  $p = D.cdf(x)$ , then  $x = D.inv(p)$ .

nonct D.nonct(cv,p) returns non-centrality value given the critical value (cv) and p value (cdf). Both cv and p could be either vector or matrix as long as the sizes are the same.

sample D.sample(), D.sample(n), and D.sample(m,n) return a random scalar or a vector of size n or a matrix of size m by n.

parameter D.parameter(1) returns  $\alpha$ , D.parameter(2) returns  $\beta$ , and D.parameter(3) returns  $\lambda$ .

mean D.mean() returns the expected value.

variance D.variance() returns the variance.

**Examples**

```
> C = StatDist("Beta",2,3)
      BetaDist(2,3,0)
> D.mean()
      0.4
> D.sample(1000).mean()
      0.407094
> D.pdf([0.01,0.5,0.9])
      Col 1      Col 2      Col 3
Row 1      0.117612      1.50000      0.108000
> D.cdf([0.01,0.5,0.9])
      Col 1      Col 2      Col 3
Row 1      0.000592030      0.687500      0.996300
> D.inv([0.000592030,0.687500,0.996300])
      Col 1      Col 2      Col 3
Row 1      0.00999928      0.500001      0.900001
```

**13.2.9 Log normal distribution****Definition**

The random variable  $X$  has a *lognormal distribution* if its probability density function (pdf) is defined by

$$f(x) = [(x - \theta)\sqrt{(2\pi)\sigma}]^{-1} \exp\left(-\frac{\log(x - \theta) - \mu)^2}{2\sigma^2}\right), \quad x > \theta \quad (13.13)$$

where  $\mu$  (real),  $\sigma^2$  (real) and  $\theta$  (real) are the parameters with their ranges  $-\infty < \mu, \theta < \infty$  and  $\sigma^2 > 0$ . In short, we say  $X \sim \text{lognormal}(\mu, \sigma^2, \theta)$ .

Alternatively, if  $\log(X - \theta) \sim N(\mu, \sigma^2)$ , then we say  $X$  has a  $\text{lognormal}(\mu, \sigma^2, \theta)$  distribution.

The lognormal distribution with  $\theta = 0$  is known as two-parameter lognormal distribution  $\text{lognormal}(\mu, \sigma^2)$ .

### Properties

1.  $E(X) = \exp(\mu + \sigma^2/2) + \theta$ ,  $\text{Var}(X) = \exp(2\mu + 2\sigma^2) - \exp(2\mu + \sigma^2) + 2\theta^2$ .
2.  $\text{mode}(X) = \exp(\mu - \sigma^2) + \theta$
3.  $\text{median}(X) = \exp(\mu) + \theta$

### Matvec interface

An object of  $\text{lognormal}(\mu, \sigma^2, \theta)$  can be created by

```
D = StatDist("LogNormal",mu,sigma2,theta);
D = StatDist("LogNormal",mu,sigma2);
```

Matvec provided several standard member functions to allow user to access most of properties and functions of  $\text{lognormal}(\mu, \sigma^2, \theta)$ :

pdf  $D.\text{pdf}(x)$  returns the probability density function (pdf) values of  $x$  which could be a vector or matrix.

cdf  $D.\text{cdf}(x)$  returns the cumulative distribution function (cdf) values of  $x$  which could be a vector or matrix

mgf  $D.\text{mgf}(t)$  returns the moment-generating function (mgf) values of  $t$  which could be a vector or matrix.

inv  $D.\text{inv}(p)$  is the inverse function of  $D.\text{cdf}(x)$ , where  $p$  could be a vector or matrix. That is if  $p = D.\text{cdf}(x)$ , then  $x = D.\text{inv}(p)$ .

sample  $D.\text{sample}()$ ,  $D.\text{sample}(n)$ , and  $D.\text{sample}(m,n)$  return a random scalar or a vector of size  $n$  or a matrix of size  $m$  by  $n$ .

parameter  $D.\text{parameter}(1)$  returns  $\mu$ ,  $D.\text{parameter}(2)$  returns  $\sigma^2$ , and  $D.\text{parameter}(3)$  returns  $\theta$ .

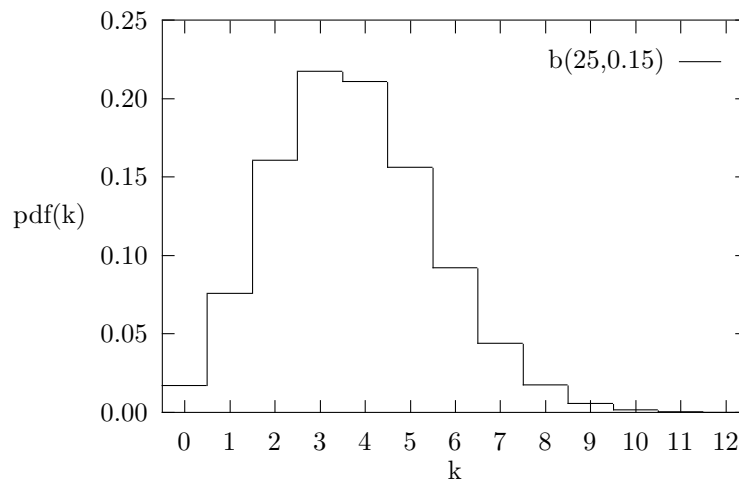
mean  $D.\text{mean}()$  returns the expected value.

variance  $D.\text{variance}()$  returns the variance.

### Examples

```
> D = StatDist("LogNormal",1,3,2);
      LogNormalDist(1,3,2)
> D.mean()
      14.1825
> D.sample(1000).mean()
      14.087
> D.pdf([2.01,10,30])
           Col 1      Col 2      Col 3
Row 1      0.122530    0.0237094    0.00332270
> D.cdf([2.01,10,30])
           Col 1      Col 2      Col 3
Row 1    0.000605776    0.733429    0.910929
> D.inv([0.000605776,0.733429,0.910929])
           Col 1      Col 2      Col 3
Row 1      2.01000    10.0000    30.0000
```

Figure 13.2: Binomial Probability Density Function (pdf)



## 13.3 Discrete Distribution

The distributions for common discrete random variables are Uniform(discrete), Binomial, Poisson, Geometric, Negative Binomial and Hypergeometric.

### 13.3.1 Binomial distribution

#### Definition

The random variable  $X$  has a *binomial distribution* if its probability density function (pdf) is defined by

$$f(x) = \frac{n!}{(n-x)!x!} p^x (1-p)^{n-x}, \quad x = 0, 1, 2, \dots, n \quad (13.14)$$

where  $p$  (real) and  $n$  (integer) are the parameters with their ranges  $0 \leq p \leq 1$ , and  $n \geq 1$ . In short, we say  $X \sim b(n, p)$ .

The graph of the pdf of  $b(25, 0.15)$  is shown in Fig. 13.2. The Matvec script used to draw this graph is on page 50.

#### Properties

1. its moment generation function is

$$M(t) = (1 - p + pe^t)^n$$

2.  $E(X) = np$ ,  $\text{Var}(X) = np(1-p)$ .

#### Matvec interface

An object of  $b(n, p)$  can be created by

```
D = StatDist("Binomial",n,p);
```

Matvec provided several standard member functions to allow user to access most of properties and functions of  $b(n, p)$ :

pdf D.pdf(x) returns the probability density function (pdf) values of  $x$  which could be a vector or matrix.

cdf `D.cdf(x)` returns the cumulative distribution function (cdf) values of  $x$  which could be a vector or matrix

mgf `D.mgf(t)` returns the moment-generating function (mgf) values of  $t$  which could be a vector or matrix.

inv `D.inv(p)` is the inverse function of `D.cdf(x)`, where  $p$  could be a vector or matrix. That is if  $p = D.cdf(x)$ , then  $x = D.inv(p)$ .

sample `D.sample()`, `D.sample(n)`, and `D.sample(m,n)` return a random scalar or a vector of size  $n$  or a matrix of size  $m$  by  $n$ .

parameter `D.parameter(1)` returns  $n$ , number of Bernoulli trials; whereas `D.parameter(2)` returns  $p$ , the probability of success in a Bernoulli trial.

mean `D.mean()` returns the expected value.

variance `D.variance()` returns the variance.

### Examples

```
> D = StatDist("Binomial",4,0.2)
      BinomialDist(4,0.2)
> D.mean()
      0.8
> D.sample(1000).mean()
      0.764
> D.pdf([0,1,4])
      Col 1      Col 2      Col 3
Row 1      0.409600      0.409600      0.00160000
> D.cdf([0,1,4])
      Col 1      Col 2      Col 3
Row 1      0.409600      0.819200      1.000000
> D.inv(0.5)

***ERROR***
BinomialDist::inv(): not available
```

## 13.3.2 Poisson distribution

### Definition

The random variable  $X$  has a *Poisson distribution* if its probability density function (pdf) is defined by

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}, \quad x = 0, 1, 2, \dots, \quad (13.15)$$

where  $\lambda$  (real) is the parameter with its range  $\lambda > 0$ . In short, we say  $X \sim P(\lambda)$ .

### Properties

1. its moment generation function is

$$M(t) = e^{\lambda(e^t - 1)}$$

2.  $E(X) = \lambda$ ,  $\text{Var}(X) = \lambda$ .



**Matvec interface**

An object of  $P(\lambda)$  can be created by

```
D = StatDist("Poisson",lambda);
```

Matvec provided several standard member functions to allow user to access most of properties and functions of  $P(\lambda)$ :

pdf D.pdf(x) returns the probability density function (pdf) values of x which could be a vector or matrix.

cdf D.cdf(x) returns the cumulative distribution function (cdf) values of x which could be a vector or matrix

mgf D.mgf(t) returns the moment-generating function (mgf) values of t which could be a vector or matrix.

inv D.inv(p) is the inverse function of D.cdf(x), where p could be a vector or matrix. That is if  $p = D.cdf(x)$ , then  $x = D.inv(p)$ .

sample D.sample(), D.sample(n), and D.sample(m,n) return a random scalar or a vector of size n or a matrix of size m by n.

parameter D.parameter(1) returns  $\lambda$ .

mean D.mean() returns the expected value.

variance D.variance() returns the variance.

**Examples**

```
> D = StatDist("Poisson",10)
      PoissonDist(10)
> D.sample(1000).mean()
      10.199
> D.sample(1000).variance()
      9.98214
> D.pdf([0,10,100])
              Col 1          Col 2          Col 3
Row 1  4.53999e-05    0.125110  4.86465e-63
> D.cdf([0,10,100])
              Col 1          Col 2          Col 3
Row 1  4.53999e-05    0.583040    1.00000
> D.inv(0.5)
```

```
***ERROR***
```

```
PoissonDist::inv(): not available
```

**13.3.3 Geometric distribution****Definition**

The random variable  $X$  has a *geometric distribution* if its probability density function (pdf) is defined by

$$f(x) = (1-p)^x p, \quad x = 0, 1, 2, \dots, \quad (13.16)$$

where  $p$  (real) is the parameter with its range  $0 \leq p \leq 1$ . In short, we say  $X \sim g(p)$ .

For example, let  $X$  be the number of failures before the first success in a sequence of Bernoulli trials with probability of success  $p$ , then  $X \sim g(p)$ .

### Properties

1. its moment generation function is

$$M(t) = \frac{p}{1 - (1-p)e^t}, \quad t < -\ln(1-p)$$

2.  $E(X) = \frac{1-p}{p}$ ,  $\text{Var}(X) = \frac{1-p}{p^2}$ .

### Matvec interface

An object of  $g(p)$  can be created by

```
D = StatDist("Geometric",p);
```

Matvec provided several standard member functions to allow user to access most of properties and functions of  $g(p)$ :

pdf D.pdf(x) returns the probability density function (pdf) values of x which could be a vector or matrix.

cdf D.cdf(x) returns the cumulative distribution function (cdf) values of x which could be a vector or matrix

mgf D.mgf(t) returns the moment-generating function (mgf) values of t which could be a vector or matrix.

inv D.inv(p) is the inverse function of D.cdf(x), where p could be a vector or matrix. That is if  $p = D.cdf(x)$ , then  $x = D.inv(p)$ .

sample D.sample(), D.sample(n), and D.sample(m,n) return a random scalar or a vector of size n or a matrix of size m by n.

parameter D.parameter(1) returns  $p$ , the probability of success in a Bernoulli trial.

mean D.mean() returns the expected value.

variance D.variance() returns the variance.

### Examples

```
> D = StatDist("Geometric",0.2)
      GeometricDist(0.2)
> D.mean()
      4
> D.sample(1000).mean()
      3.751
> D.pdf([1,2,10])
      Col 1      Col 2      Col 3
Row 1      0.160000      0.128000      0.0214748
> D.cdf([1,2,10])
      Col 1      Col 2      Col 3
Row 1      0.360000      0.488000      0.914101
> D.inv(0.5)
```

```
***ERROR***
```

```
GeometricDist::inv(): not available
```

### 13.3.4 Negative binomial distribution

#### Definition

The random variable  $X$  has a *negative binomial distribution* if its probability density function (pdf) is defined by

$$f(x) = \frac{x+n-1}{x!(n-1)!} p^n (1-p)^x, \quad x = 0, 1, 2, \dots, \quad (13.17)$$

where  $p$  (real) and  $n$  (integer) are the parameters with their ranges  $0 \leq p \leq 1$  and  $n \geq 1$ .

For example, let  $X$  be the number of failures before the  $n$ 'th success in a sequence of Bernoulli trials with probability of success  $p$ , then  $X$  is distributed as negative binomial distribution with parameters  $n$  and  $p$ .

#### Properties

1. its moment generation function is

$$M(t) = \frac{p^r}{[1 - (1-p)e^t]^r}, \quad t < -\ln(1-p)$$

2.  $E(X) = \frac{r(1-p)}{p}$ ,  $\text{Var}(X) = \frac{r(1-p)}{p^2}$ .

#### Matvec interface

An object of the negative binomial distribution can be created by

```
D = StatDist("NegativeBinomial",n,p);
```

Matvec provided several standard member functions to allow user to access most of properties and functions of the negative binomial distribution:

pdf `D.pdf(x)` returns the probability density function (pdf) values of  $x$  which could be a vector or matrix.

cdf `D.cdf(x)` returns the cumulative distribution function (cdf) values of  $x$  which could be a vector or matrix

mgf `D.mgf(t)` returns the moment-generating function (mgf) values of  $t$  which could be a vector or matrix.

inv `D.inv(p)` is the inverse function of `D.cdf(x)`, where  $p$  could be a vector or matrix. That is if  $p = D.cdf(x)$ , then  $x = D.inv(p)$ .

sample `D.sample()`, `D.sample(n)`, and `D.sample(m,n)` return a random scalar or a vector of size  $n$  or a matrix of size  $m$  by  $n$ .

parameter `D.parameter(1)` returns  $n$ , number of Bernoulli trials; whereas `D.parameter(2)` returns  $p$ , the probability of success in a Bernoulli trial.

mean `D.mean()` returns the expected value.

variance `D.variance()` returns the variance.

#### Examples

```
> D = StatDist("NegativeBinomial",4,0.1)
      NegativeBinomialDist(4,0.1)
> D.mean()
      36
> D.sample(1000).mean()
      36.256
> D.pdf([0,10,100])
```

```
      Col 1      Col 2      Col 3
Row 1 0.000100000 0.00997220 0.000469741
> D.cdf([0,10,100])
```

```
      Col 1      Col 2      Col 3
Row 1 0.000100000 0.0441329 0.994276
> D.inv(0.5)
```

\*\*\*ERROR\*\*\*

NegativeBinomialDist::inv(): not available

# Chapter 14

## Exception

Exceptions in Matvec

### 14.1 Error Handling

Matvec does not handle errors very gracefully. You will be forced to quit Matvec if the number of errors reaches 15 in a session.

After numerous errors, you may find that Matvec behaves unexpectedly, then you REALLY have to quit the current session immediately and restart it again.

### 14.2 Troubleshooting

#### 14.2.1 Lack of memory

Because Matvec allocate memory dynamically for each object. If you get a message saying “lack of memory”, then you have reached the maximum memory in your computer.

#### 14.2.2 Intermediate files

Matvec may automatically swap data from the computer memory to disk. These intermediate files are saved in the `$(HOME)/.matvec` directory with file names something like `798233981.2` where `798233981` is your computer system time when you launched Matvec, and `2` indicates that this is the second intermediate file since you launched Matvec. It is clear that you are safe to launch multiple Matvec sessions within the same directory as long as those sessions started at different time. All of these intermediate files are automatically deleted whenever Matvec is normally ended; otherwise you have to manually delete them.

#### 14.2.3 Bugs

- Whenever you get a message similar to “you have probably found a bug!”, report to Matvec maintainer with your script and data.
- Whenever you suspect a bug, don’t report it until you can duplicate the bug.

#### 14.2.4 EPSILON

- if you get a message similar to

```
***ERROR***  
SparseMatrix seems not positive (semi)definite: -3284690426.413546
```

```
***ERROR***  
in sdrvd(), nonzero flag 1
```

Then you have to increase EPSILON value (the default is 1.0e-14) using

```
this.parameter("EPSILON",1.0e-13);
```

or even larger.

### 14.3 Limitation

There are almost no hard-code in Matvec implementation. Thus, there are almost no limitations except the following one.

- One data record must be within one line. Multi-line data record is not allowed. This limitation may be removed later.

## Chapter 15

# Matvec C++ API

### 15.1 An Overview of C++ Class Library

### 15.2 Examples





# Index

- `*`, 16
  - `*=`, 16
  - `+`, 16
  - `++`, 16
  - `+=`, 16
  - `-`, 16
  - `-`, 16
  - `-=`, 16
  - `.*`, 16
  - `.*=`, 16
  - `.+`, 16
  - `.+=`, 16
  - `.-`, 16
  - `.-=`, 16
  - `./`, 16
  - `./=`, 16
  - `.^`, 16
  - `/`, 16
  - `/=`, 16
  - `=`, 16
  - `==`, 17
  - `^`, 16
  - `'`, 16
  - `<=`, 17
  - `<`, 17
  - `>=`, 17
  - `>`, 17
- AI REML, 79
- all, 19
- anterior probability, 87
- any, 20
- asctime, 42
- @, 16
- beta distribution, 100
  - cdf, 101
  - inv, 101
  - mean, 101
  - mgf, 101
  - parameter, 101
  - pdf, 101
  - sample, 101
  - variance, 101
- binomial distribution, 103
  - cdf, 104
  - inv, 104
  - mean, 104
  - mgf, 104
  - parameter, 104
  - pdf, 103
  - sample, 104
  - variance, 104
- BLUE, 59
- BLUP, 59
- Chi square distribution, 93
  - cdf, 94
  - inv, 94
  - mean, 94
  - mgf, 94
  - parameter, 94
  - pdf, 94
  - sample, 94
  - variance, 94
- chol, 20
- Cholesky, 20
- clock, 42
- cond, 20
- continuation, 6
- continuous distribution, 90
- contrast, 64, 72
- corrcoef, 20
- correlated random effects, 80
  - multivariate example, 81
- covariance, 21
- ctime, 42
- Data, **13**
- date, 42
- det, 21
- DF\_REML, 66
- diag, 21
- difftime, 42
- discrete distribution, 103
- distribution
  - beta, 100
  - binomial, 103
  - Chi square, 93
  - exponential, 99

- F, 96
- gamma, 98
- geometric, 105
- lognormal, 101
- negative binomial, 107
- non-central beta, 100
- non-central Chi square, 93
- non-central F, 96
- non-central t, 95
- normal, 90
- Poisson, 104
- t, 95
- uniform, 92
  
- eigen, 22
- eigenvalues, 22
- eigenvectors, 22
- else, 35
- Elston, 87
- EM.REML, 66
- EPSILON, 9
- escape sequence, 11
- estimation, 64, 71
- Exception, **109**
- !, 16
- !=, 17
- exponential distribution, 99
  - cdf, 100
  - inv, 100
  - mean, 100
  - mgf, 100
  - parameter, 100
  - pdf, 99
  - sample, 100
  - variance, 100
  
- F distribution, 96
  - cdf, 97
  - inv, 97
  - mean, 97
  - mgf, 97
  - parameter, 97
  - pdf, 97
  - sample, 97
  - variance, 97
- Fernando, 87
- fliplr, 22
- flipud, 23
- for, 36
  
- gamma distribution, 98
  - cdf, 98
  - inv, 98
  - mean, 98
  - mgf, 98
  - parameter, 98
  - pdf, 98
  - sample, 98
  - variance, 98
- gammaminc, 19
- gammaln, 19
- genotype probability, 87
- geometric distribution, 105
  - cdf, 106
  - inv, 106
  - mean, 106
  - mgf, 106
  - parameter, 106
  - pdf, 106
  - sample, 106
  - variance, 106
- ginv, 23
- glim, 71
- GLMM, **69**
  - AI REML, 79
  - contrast, 72
  - estimation, 71
  - glim, 71
  - GLM, 69
  - growth curves, 77
  - hypothesis test, 72
  - linked variance functions, 84
  - Model Specification, 70
  - survival, 74
  - threshold model, 73
  - variance components estimation, 79
- gmtime, 42
- Growth curves
  - Richards family, 77
  
- hadamard, 56
- hankel, 55
- Heuch, 87
- hilb, 55
- hypothesis test, 64, 72
  
- identity, 24
- if, 35
- input, 24
- INPUT\_LINE\_WIDTH, 9
- inv, 24
- invhilb, 55
- iterative algorithm, 87
  
- Janss, 87
  
- kron, 24
- Kronecker, 24

vander, 56

- variance, 31
- vec, 31
- vech, 32
- zeros, 32
- max, 25
- MAX\_ERRORS, 9
- mean, 25
- min, 25
- mktime, 42
- mod, 26
- multi-trait, 63
- Murphy, 87
- Mutalik, 87
- negative binomial distribution, 107
  - cdf, 107
  - inv, 107
  - mean, 107
  - mgf, 107
  - parameter, 107
  - pdf, 107
  - sample, 107
  - variance, 107
- non-central beta distribution, 100
  - nonct, 101
- non-central Chi square distribution, 93
  - nonct, 94
- non-central F distribution, 96
  - nonct, 97
- non-central t distribution, 95
  - nonct, 96
- norm, 26
- normal distribution, 90
  - cdf, 91
  - inv, 91
  - mean, 91
  - mgf, 91
  - parameter, 91
  - pdf, 91
  - sample, 91
  - variance, 91
- Objects, 9
- ones, 26
- OUTPUT\_LINE\_WIDTH, 9
- OUTPUT\_PRECISION, 9
- pascal, 56
- Pedigree, **12**
- Poisson distribution, 104
  - cdf, 105
  - inv, 105
  - mean, 105
  - mgf, 105
  - parameter, 105
  - pdf, 105
  - sample, 105
  - variance, 105
- posterior probability, 87
- pow, 26
- product, 26
- Program Flow Control, **35**
  - else, 35
  - for, 36
  - if, 35
  - repeat, 37
  - until, 37
  - while, 37
- rank, 26
- regression, 59
- REML, 66
- repeat, 37
- reshape, 27
- resize, 27
- rot90, 27
- sample, 27
- save, 28
- select, 28
- singular value decomposition, 29
- solve, 28
- sort, 29
- splines, 28
  - mixed model example, 84
- Statistical Distributions, **89**
  - cdf, 89
  - inv, 89
  - mean, 89
  - mgf, 89
  - nonct, 89
  - parameter, 89
  - pdf, 89
  - random, 89
  - sample, 89
  - StatDist, 89
  - variance, 89
- std, 29
- Stewart, 87
- String, **11**
  - escape sequence, 11
- sum, 29
- sumsq, 29
- survival, 74
- svd, 29
- sweep, 30
- t, 30

- t distribution, 95
  - cdf, 96
  - inv, 96
  - mean, 96
  - mgf, 96
  - parameter, 96
  - pdf, 95
  - sample, 96
  - variance, 96
- threshold, 73
- time, 41
- Time Control, **41**
  - asctime, 42
  - clock, 42
  - ctime, 42
  - date, 42
  - difftime, 42
  - gmtime, 42
  - leapyear, 43
  - localtime, 41
  - mktime, 42
  - time, 41
- toeplitz, 56
- trace, 30
- transpose, 16
- tril, 30
- triu, 31
- uniform distribution, 92
  - cdf, 92
  - inv, 92
  - mean, 92
  - mgf, 92
  - parameter, 92
  - pdf, 92
  - sample, 92
  - variance, 92
- until, 37
- Van Arendonk, 87
- vander, 56
- variance, 31
- variance components estimation, 66, 79
- vec, 31
- vech, 32
- Vector, **10**
  - ==, 17
  - !=, 17
  - <=, 17
  - <, 17
  - >=, 17
  - >, 17
- Wang, 87
- WARNING, 9
- while, 37
- zeros, 32