GAUSS Programming for Econometricians

Soosung Hwang[1]
Dept. of Banking and Finance,
City University Business School

September 25, 2001

[1]Address: Faculty of Finance, City University Business School, Frobisher Crescent, Barbican Centre, London EC2Y 8HB, UK. Email: s.hwang@city.ac.uk, Tel: +44 (0)20 7477 0109.

Abstract

Many universities and research institutions have used GAUSS in their econometric curriculum. Unfortunately, GAUSS is not an easy language to learn and master, particularly for those without computer programming experience.

This is an introductory manual, which only covers the most fundamental parts of GAUSS and thus is supplementary to the official GAUSS manual. However, I tried to make this text readable as a stand-alone manual with an exercise GAUSS code.

The purpose of this course is on acquiring familiarity with the fundamentals of GAUSS and programming competence. Once the basics of programming in GAUSS have been mastered, then the competent user can make use of various range of procedures and functions in addition to those included in GAUSS standard package. They are both commercial and non-commercial applications which can be accessed easily via internet.

GAUSS package has been developed continuously and there are many different versions. However, the material difference between versions are relatively small and I do not expect any difficult situations from using different versions of GAUSS.

Remember that you can not learn GAUSS (and also econometrics) by "just" reading manual. You must interact with the computer and manual by working through the examples.

1. Introduction

1.1 What is GAUSS?

GAUSS is a **high level matrix programming language** specializing in commands, functions, and procedures for data analysis and statistical applications. It is a general purpose tool; it is not a specialised econometric packages such as STAMP. GAUSS is a programming language similar to C or Pascal.

GAUSS is a programming language designed for matrix-based operations and manipulations, suitable for high level statistical and econometric computation. Although many universities and research institutions have used GAUSS in their econometric curriculum, it is not easy to learn and master GAUSS, particularly for those without computer programming experience.

Gauss functions can be roughly divided into three categories:

- The **Mathematical, statistical and scientific Functions** are those to be expected in a high level mathematical language: trigonometric functions, distribution functions, random number generators, numerical differentiation and integration routines, Fourier transforms, Bessel functions and polynomial evaluation routines. In addition, as a matrix programming language, GAUSS includes a variety of routines which perform standard matrix operations. Among these are routines to calculate determinants, matrix inverses, decompositions, eigenvalues and eigenvectors.
- The **Data Handling routines** (including data matrix manipulation and description routines, and file I/O) include functions which return dimensions of matrices, and information about elements of data matrices, including functions to locate values lying in specific ranges or with certain values. Also under data handling routines fall all those functions which create, save, open and read from and write to GAUSS data sets.
- The **Programming statements** (including branching, looping, display features, error checking, and shell commands) are all of the commands which make it possible to write complex programmes in GAUSS. These include conditional and unconditional branching, looping, file I/O, error handling, and system-related commands to execute shells and access directory and environment information.

1.2 Advantages and Disadvantages of GAUSS

Advantages

- **Flexibility**; GAUSS is very flexible for a wider range of applications than standard econometric packages because it is a general programming language. GAUSS programmes and functions are all available to the user, and so the user is able to change them. Similarly, if data is held in a non-standard format, you may write your own routine to access it.

- **Fast and Efficient**; GAUSS is fast and efficient because it operates directly on matrices. This makes it more useful for economists than standard programming languages where the basic data units are all scalars.

Disadvantages

- **High Costs**; the costs of using GAUSS are high. Its high flexibility means that there is unlikely to be a simple procedure to do a simple econometric task.
- **Not Easy to Learn**; even if pre-programmed commercial or non-commercial GAUSS codes are available for a task, a reasonable degree of familiarity with GAUSS and its methods will often be necessary to make effective use of such routines.
- **High Possibility of Mistakes**; GAUSS is too tolerant of sloppy programming and errors. It is difficult for the computer or user to detect errors.

1.3 When to use GAUSS

GAUSS is ideally suited to **non-standard tasks**. When you may wish to vary or add to standard techniques, GAUSS is an excellent alternative. To get extra specification tests, for example, a straightforward solution would be to code a routine and emend the preexisting GAUSS programme to call the new procedure at the appropriate point. Note that GAUSS is not appropriate for standard applications on standard datasets; there is little point in writing an OLS estimation routine in GAUSS for a dataset. In this case, you can use routines commercially available for the OLS. Many statistical software packages such as SAS, RATS, TSP, LimDep, etc., will perform the estimation and there is no necessity to learn anything at all about GAUSS to use these programmes.

If the core of your task is **matrix manipulation**, then GAUSS is likely to be a better bet than a full programming language.

There are many ready-made GAUSS codes in the Web site, which you can easily access. If the codes are not developed by a commercial company, they are free. The followings are some of the GAUSS web sites.

http://www.timberlake.co.uk/

http://gurukul.ucc.american.edu/econ/gaussres/GAUSSIDX.HTM

gopher://mundo.eco.utexas.edu/1m/mailing/gaussians.archive

2. GAUSS BASICS

Before we start, here are some useful but important rules of syntax in GAUSS code.

- GAUSS is not case-sensitive.
- Comments can be written with @ ... @ or /* ... */.
- Each line of code must end with a semi-colon, " ; ".
- The equal sign '=' is the assignment operator.

$$y = x + z;$$

- GAUSS command print is used to print output to the screen.
- A GAUSS programme must end with the statement "end;".

2.1 Data Input

GAUSS reads input from, and writes output to, a number of types of file. The important three kinds of data format are GAUSS data file, GAUSS matrix file, and text (or ASCII) file. The first two data file formats are unique and efficient in GAUSS. For file transfer (import and export) between GAUSS and other application software or across platforms, the text file format is preferred.

Here we explain the following three methods; ASCII files, Microsoft excel spreadsheet, and GAUSS datasets (idea only). We consider each of these in turn, starting with the simplest.

2.1.1 Typing

The method is not usually used in economics and finance, since the number of observations is too large to type all of them one by one. However, in some situation, it is useful.

If you want to create 3×1 column vector, type

$$x = \{\ 1,\ \ 2,\ \ 3\ \};$$

and GAUSS reads

$$x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

Using the same method, if you type

$$x = (\ 1\ \ \ 2\ \ \ 3\ );$$

GAUSS recognises it as the following 1×3 row vector

$$x = (\ 1\ \ \ 2\ \ \ 3\ ).$$

Remember that 'comma' is used to create rows.

Therefore, to create a (3×3) matrix, type the following

$$x = \{\ 1 \quad 2 \quad 3, \quad 4 \quad 5 \quad 6, \quad 7 \quad 8 \quad 9\ \};$$

and Gauss reads it as

$$x = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

2.1.2 ASCII Files

GAUSS can read ASCII files which most other software can also read. Therefore, it is possible to share data between GAUSS and other programmes. ASCII files can be created in any spread sheet programme such as Excel or obtained from data providers such as Datastreem.

The command to read ASCII files is

$$\begin{array}{rrcl} & load\ variablename & = & filename \\ \text{e.g.} & load\ x[m,n] & = & c:\backslash lec-gaus\backslash ter1.txt; \end{array}$$

where x is the name of the matrix assigned to the matrix, m and n are numbers of rows and columns, '$c:\backslash lec-gaus$' is an example of path, and '$ter1.txt$' is the name of the ASCII file in the directory of '$c:\backslash lec-gaus$'. Note the following two points.

- The matrix name x can be any character; e.g., xx, y, yy, $y1$, $x1$, etc.
- You have to know the exact values for m and n.

2.1.3 GAUSS Data Files

GAUSS datasets are created by writing data from GAUSS or by taking an ASCII file and converting through a stand-alone programme called ATOG.EXE (ASCII TO Gauss). As with the datasets for other econometric packages, they consist of rows of data split into fields. The actual dataset is held in the .dat (data) file, while the .dht (header) file contains the names of each of these fields, along with some other information about the data file. GAUSS will automatically add .dat (or .dht) to the filenames you give, and so there is no need to include the extension.

Unlike the GAUSS matrices, reading from or writing to a GAUSS dataset is not a single, simple operation. For matrices, the whole object is being moved into memory or onto disk. By contrast, a GAUSS dataset is used in a number of stages. Firstly, the file must be opened; then it may be read from or written to, which may involve the whole file or just a few lines; finally, when references to the file are finished, it should be closed.

In this course we do not discuss how to create GAUSS data file in detail, since we may not detect any difference in data reading speeds between ASCII files and GAUSS data files due to recent computer hardware development. For detail explanations on how to create GAUSS data, see CREATE in the COMMAND REFERENCE for details.

Once we create GAUSS data, we use the following statement to read them.

$$open\ x = data;$$

where x is a matrix name and *data* is the GAUSS data set.

2.1.4 Spread Sheet Files

GAUSS for Windows NT/95 Version allows us to import data from a spread sheet to a GAUSS matrix or data set. It takes relatively long time to load dataset from a spread sheet. The following file types are supported: "WKS" "WK1" "WK2" (Lotus v1-v2), "WK3" "WK4" "WK5" (Lotus v3-v5), "XLS" (Excel v2.1-v7.0), "WQ1" "WQ2" "WB1" (Quattro v1-v6), "WRK" (Symphony v1.0-1.1), "DB2" (dBase II), "DBF" (dBase III/IV), Foxpro, Clipper, "DB" (Paradox), "CSV" "TXT" "ASC" (ASCII character delimited), "PRN" (ASCII packed), and "DAT" (GAUSS data set).

The command to read spread sheet files is

$$\{x, namelist\} = import(fname, range, sheet);$$

where x is a matrix name assigned to the matrix read by GAUSS, *namelist* is a character vector of column names, $fname$ is path and file name of source file, *range* is range of cells for spreadsheets, *sheet* is a scalar for page or sheet number.

For example,

$$\begin{aligned} fname &= "C:\backslash\backslash lec-gaus\backslash\backslash autocorr.xls"; \\ range &= "a7:e116"; \\ \{x, names\} &= import(fname, range, 1); \end{aligned}$$

This will read the data in the range from $a7 : e116$ in the second spreadsheet in *autocorr.xls* file.

2.2 Output

This subsection explains how to write output to an ASCII file.

PRINT command is used to print the output. For example, when you type

$$\begin{aligned} x &= \{\ 1\ \ 2\ \ 3,\ \ 4\ \ 5\ \ 6,\ \ 7\ \ 8\ \ 9\ \}; \\ & print\ "x ="\ x; \end{aligned}$$

we have

$$x = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}.$$

Here any words between the quotation marks "$x =$" are printed as string constants.

However, PRINT command itself is not enough to edit with another text editor; the output is not saved in hard disk. The out files should be saved as an ASCII file

and available to other text editor or programmes. To do this we add the following command line

$$\begin{aligned} & format\ /rd\ 12,10; \\ & outwidth\ 256; \\ output\ file\ = & \ C:\backslash lec-gaus\backslash lec-gaus.out\ reset; \end{aligned}$$

format /rd 12,10	a field width of 12 spaces per number and with 10 places beyound the decimal point
outwidth 256	maximum column line

2.3 Variables

2.3.1 Variable Types

There are two types of GAUSS variables; matrices and strings. Matrices include vectors and scalars, but these are treated the same by GAUSS. **Matrices** can have either numeric or character elements or both. Numerical data are stored in scientific notation to around 12 places of precision with a range of about $10^{\pm 35}$. Character data are sequences of characters which count as one element of the matrix.

Strings are pieces of text of unlimited length. These are used to give information to the user. They can be used to store the names of files to be opened, messages to be printed, entries files, etc.

2.3.2 Creation of Variable

All variables must be created and given an initial value before they are referenced. Acceptable names for variables can contain alphanumeric data and the underscore, but must not begin with a number. Reserved words may not be used.

Acceptable variable names:

x xx xy yyy $yyyy$ $xxxy$

Unacceptable variable names:

100 if (reserved word) $sqrt$ (reserved word)

2.3.3 Creating matrices

New matrices can be defined at any point. The easiest way is to assign a value to one. There are two ways to do this - by assigning a constant value or by assigning the result of some operation. See input subsection.

3. MATRIX ALGEBRA AND MANIPULATION

3.1 Special Matrix Functions

To create a matrix of zeros or ones, or random numbers more easily, we can use the following statements.

Matrix Types	Statement
Zeros	$x = zeros(4, 3);$
Ones	$Ones\_matrix = ones(8, 4);$
Uniform Random Numbers	$z = rndu(7, 4);$
Normal Random Numbers	$xx = rndn(10, 4);$
Identity Matrix	$identity\_matrix = eye(3);$

3.2 Element-by-Element Operators

GAUSS allows all of the mathematical and logical operators to be prefixed by a dot. This tells the machine that operations are to be carried out on an "element by element" basis; the operands are essentially broken down into the smallest conformable elements and then the scalar operators are applied. However, for addition and subtraction, we do not need a dot and they are

$$z = x + y; z = x - y;$$

How this works in practice depends on the matrices. The operators described as element-by-element operators have rules of conformability. To give an example, in a typical expression involving an element-by-element operator

$$z = x + y;$$

conformability is defined as follows;

Cases	x	y	z
1	1 8 4 5 2 9 7 6 3	1 5 8 6 7 4 4 5 3	2 13 12 11 9 13 11 11 6
2	1 8 4 5 2 9 7 6 3	2	3 10 6 7 4 11 9 8 5
3	1 8 4 5 2 9 7 6 3	2 7 4	3 10 6 12 9 16 11 10 7
4	1 8 4 5 2 9 7 6 3	3 9 6	4 17 10 8 11 15 10 15 9
5	2 7 4	3 9 6	5 11 8 10 16 13 7 13 10

If x and y are such that none of these conditions apply, then the matrices are not conformable to these operations and an error message will appear.

The element-by-element (EBE) operators are

Addition	+
Subtraction	-
Element-by-Emelemt Multiplication	.*
Element-by-Emelemt Division	./
Exponentiation	^or .^

3.3 Matrix Operators

GAUSS generally operates in the usual way. If a scalar operand is applied to a matrix, then the operation will be applied to every element of the matrix. If two matrices are involved, the usual conformability rules apply. Note that the conformability in matrix algebra is not the same as that in element-by-element operators (for addition and subtraction, see the above EBE operators). Conformability rules for operators are (z=x operator y)

Operation		x	y	x
Multiplication	*	$M \times N$	$N \times L$	$M \times L$
Division	/	Matrix	scalar	EBE division
		Matrix	scalar	EBE division
		$M \times N$	Square matrix$_{(N \times N)}$	$x * y^{-1}$
		$M \times N$	Rectangular matrix$_{(N \times L)}$	$(x'x)^{-1}x'y$

Other useful operators (e.g., $x \tilde{} y = z$)are

Operation		x	y	z
Kronecker product	.*.	x_{11} x_{12} x_{21} x_{22}	y_1 y_2	$x_{11}y_1$ $x_{11}y_2$ $x_{12}y_1$ $x_{12}y_2$ $x_{21}y_1$ $x_{21}y_2$ $x_{22}y_1$ $x_{22}y_2$
Horizontal Concatenation	~	x_{11} x_{12} x_{21} x_{22}	y_{11} y_{12} y_{21} y_{22}	x_{11} x_{12} y_{11} y_{12} x_{21} x_{22} y_{21} y_{22}
Vertical Concatenation	|	x_1 x_2	y_1 y_2	x_1 x_2 y_1 y_2

In addition, we have (e.g., $x! = z$)

Operation		x	z
Transposition	'	x_{11} x_{12} x_{21} x_{22}	x_{11} x_{21} x_{12} x_{22}
Factorial	!	x_{11} x_{12} x_{21} x_{22}	$x_{11}!$ $x_{12}!$ $x_{21}!$ $x_{22}!$

3.4 Relational Operators

We have six relational operators; these are ==, /=, >, <, >=, <=.

- If the relational operator is not preceded by a dot, then the result is always a scalar 1 or 0, based upon a comparison of all elements of x and y.
- If the relational operator is preceded by a dot, then the result will be a matrix of 1's and 0's, based upon an element-by-element comparison of x and y.

Relations	Matrix Relations	EBE Relations
Less than	$x < y;$ $x\ LT\ y;$ $x\ \$ <\ y;$	$x\ . < y;$ $x\ .LT\ y;$ $x\ .\$ <\ y;$
less than or equal to	$x < -y;$ $x\ LE\ y;$ $x\ \$ <= \ y;$	$x\ . < -y;$ $x\ .LE\ y;$ $x\ .\$ <= \ y;$
equal to	$x == y;$ $x\ EQ\ y;$ $x\ \$ ==\ y;$	$x\ . == y;$ $x\ .EQ\ y;$ $x\ .\$ ==\ y;$
not equal	$x/ = y;$ $x\ NE\ y;$ $x\ \$/ =\ y;$	$x\ ./ = y;$ $x\ .NE\ y;$ $x\ .\$/ =\ y;$
greater than or equal to	$x >= y;$ $x\ GE\ y;$ $x\ \$ >=\ y;$	$x\ . >= y;$ $x\ .GE\ y;$ $x\ .\$ >=\ y;$
greater than	$x > y;$ $x\ GT\ y;$ $x\ \$ >\ y;$	$x\ . > y;$ $x\ .GT\ y;$ $x\ .\$ >\ y;$

3.5 Other Operators

Operators	Example	Comments
Assignment	$y = 3;$	
Comma	$clear\ x, y, z;$ $y = x[3, 5];$ $y = sortc(x, c);$	- delimit lists - separate row indices from column indices within brackets - separate arguments of functions within parentheses
Period	$y = x[., 5];$ $y = x[5, .];$	- Dots in brackets signify all rows. - Dots in brackets signify all columns.
Space	$y = x[1\ 2\ 3, 2\ 5\ 8];$	- separate indices
Colon	$y = x[1 : 5, .];$	- to create a continuous range of indices

4. PROGRAMME CONTROL

Up to now all the codes in the examples and exercises are more or less definitions and how to calculate matrix. This section considers decision making and looping to control the order in which computations are done. We explain two flow control statements.

4.1 Looping

To control looping we use the DO statement in GAUSS. The loop stops repeating itself when some condition is met. When the condition is met, the programme leaps the loop and continues executing after the loop code.

Statements	Example	Comments
DO UNTIL *expression*; . *statements*; . ENDO;	$sum = 0;$ $t = 1;$ $do\ until\ t. > 50;$ $sum = sum + t;$ $t = t + 1;$ $endo;$ $print\ sum;$	The example calculates sum of 1 to 50.

4.2 Conditional Branching

A conditional branching operation would be useful in a general context; do something IF some condition is true do this otherwise do that END branching operation.

Statements	Example	Comments
IF *expression*; . *statements*; . ENDIF;	$x = rndu(100, 1);$ $if\ x <= 1;$ $print\ "This\ is\ true.";$ $endif;$	The uniform random number matrix (100×1) generated is always less than or equal to 1. We expect "This is true.".

Now we turn to a case which is a mixture of looping and conditional branching.

Statements	Example	Comments
IF *expression*; *statements*; ELSEIF *expression*; *statements*; ELSEIF *expression*; *statements*; . ENDIF;	$x = rndn(100000, 1);$ $sd1 = 0; sd2 = 0;$ $t = 1;$ $do\ until\ t. > rows(x);$ $if\ abs(x[t]) < 1;$ $sd1 = sd1 + 1;$ $elseif\ abs(x[t]) < 2;$ $sd2 = sd2 + 1;$ $endif;$ $t = t + 1;$ $endo;$ $print\ sd1./100000\ sd2./100000;$	- generates the standard normal random number matrix (100000×1). - calculates the numbers of cases within 1 STD and 2 STDs.

In many cases, looping and conditional branching need many repeated calculations, which requires a lot of calculation time. An alternative way to avoid this problem is to use matrix as follows;

$x = rndn(100000, 1);$
$z1 = abs(x). < 1;$
$z2 = abs(x). < 2;$
$sd1 = sumc(z1);$
$sd2 = sumc(z2) - sd1;$
$print\ sd1./100000\ sd2./100000;$

GAUSS also provides a useful unconditional branching called GOTO and GOSUB. Here is an example. Note that we need colon instead of semi-colon at the end of label.

Statements	Example	Comments
$label:$. $goto;$	$sum = 0;$ $t = 1;$ $restart:$ $if\ t. <= 50;$ $sum = sum + t;$ $t = t + 1;$ $goto\ restart;$ $endif;$ $print\ sum;$	The example calculates sum of 1 to 50.

5. PROCEDURES

In many cases we face a large computing task which needs hundreds of lines, sometimes thousands of lines. An excessively large and complicated programme may be difficult to read, understand, and alter. In addition, some functions may be used in many places. In these cases we need to use PROCEDURE.

Procedures are short self-contained blocks of code for smaller tasks. When they are called by the programme, the chain of command within the programme switches to the procedure; when the procedure has completed all its operations, control returns to the main programme. This section discusses how procedures are written and work.

5.1 Defining a Procedure

A procedure consists of five parts, four of which are denoted by explicit GAUSS commands;

Statements	Explanations	Example
PROC statement	Procedure name	$proc\ (3) = regress(x, y);$
LOCAL statement	Local variable	$local\ b, xx, e, sse, sd, t;$
	Body of procedure	$xx = ones(rows(x), 1)\tilde{}\, x;$
		$b = inv(xx'xx) * (xx'y);$
		$e = y - xx * b;$
		$sse = e'e/(rows(x) - cols(x));$
		$sd = sqrt(diag(sse * inv(x'x)));$
		$t = b./sd;$
RETP statement	Return of procedure	$RETP(b, sd, t);$
EDNP statement	End of procedure	$ENDP;$

Now let us discuss the five parts of the procedure definition in more detail.

- Procedure declaration:
  - PROC (#)=*name*(*argument names*);
  - #: scalar number, number of values returned by the procedure
  - *name*: procedure name
  - *arguement names*: names used inside of the procedure for the arguments that are passed to the procedure.
- Local variable declaration:
  - LOCAL x, y, a, b, ee, sse;
  - Local variables are variables known only to the procedure being defined.
- Body of procedure

  - This part includes any GAUSS statements necessary to perform the task the procedure is being written for. User defined functions and other procedures can be referenced as well as any global matrix and strings.

- Return from the procedure
  - RETP(a, b, c, x, y);
  - The RETP statement can have multiple arguments
  - The number of items returns must equal to the value of # in the PROC statement.

- End of procedure definition
  - ENDP;

5.2 How to Use a Procedure

We have a few methods to use the procedure in our GAUSS code.

- Print all returns of the procedure

$$print\ regress(x, y);$$

- Get return and save them in the names of a, b, and c. Then modify the returns in an appropriate method.

$$\begin{aligned} \{a, b, c\} &= regress(x, y); \\ & \quad print\ a\tilde{}b\tilde{}c; \end{aligned}$$

- Call a function or procedure when the return value is not needed.

$$\begin{aligned} & call\ chol(x); \\ y &= detl; \end{aligned}$$

  - The above example is the fastest way to compute the determinant of a positive definite matrix. The result of $CHOL$ (Cholesky decomposition) is discarded and $DETL$ is used to retrieve the determinant that was computed during the call to $CHOL$.

6. LIBRARIES

GAUSS uses many ready-made and user defined functions. The GAUSS library system allows for the creation and maintenance of these modular programmes. A GAUSS library is a text file that serves as a dictionary to the source files that contain the symbol definitions. A library file may contain several procedures that are all related. Note that the procedures (Usually the procedure files have an .SRC extension.) should be already created and located in an appropriate directory (Usually in the SRC directory). On the other hand library files exist in the LIB directory and have an .LCG extension.

The first place GAUSS looks for a symbol definition is in the active libraries. Libraries are activated by the command 'LIBRARY', e.g.,

library maxlik;

See LIB in the command reference for the explanation of how to make a library file.

Note that once activated via the library, the file containing procedure definition everything in that file will be compiled.

6.1 Forward References

You can create libraries of frequently used functions that the GAUSS system will automatically find and compile whenever they are referenced in a programme.

When the compiler encounters a symbol that has not previously been defined, that is called a forward reference. GAUSS handles forward references in two ways, depending on whether they are left-hand side or right-hand side references.

A left-hand side reference is usually a reference to a symbol on the left-hand side of the equal sign in an expression. A right-hand side reference is usually a reference to a symbol on the right-hand side of the equal sign in an expression.

The autoloader resolves references to procedures, keywords, matrices, and strings that are not defined in the programme from which they are referenced. The autoloader automatically locates and compiles the files containing the symbol definitions that are not resolved during the compilation of the main file.

- When the autoloader is OFF, no forward references are allowed. An external statement can be used above the first reference to a symbol, but the definition of the symbol must be in the main file or in one of the files that are #INCLUDE.
- When the autoloader is ON, GAUSS searches for unsolved symbol references during compilation using a specific search path.

7. GRAPHICS

GAUSS Graphics is a set of routines built on the graphics functions. The main graphics routines include xy, xyz, surface, polar and log plots as well as histograms, bar, and box graphs. Users can enhance their graphs by adding legends, changing fonts, and adding extra lines, arrows, symbols and messages.

In this course, however, we do not discuss about the GAUSS graphics in detail. This is because other spread sheet programmes such as the Microsoft Excel are much more powerful in graphics. Once we obtain results and save them in ASCII files, then other powerful tools can be used for graphing.

7.1 Configuration

The computer's hardware configuration for the Publication Quality Graphics system can be supplied by the user. The configuration file called PQGRUN.CFG located in the GAUSS directory contains parameters which reflect computer's display, printer, and file format information. The configuration file is a self-documented version.

7.2 Using GAUSS Graphics

There are four basic parts to a graphics programme. These elements should be in any programmes that uses graphics routines.

- Header: To use the praphics procedures, the PGRAPH library must be active. You have to put the LIBRARY statement at the top of your programme or command file. The next line will be a command to reset the praphics global variables to the default state.
- Data Setup: The data to be graphed must be in matrices.
- Graphics Format Setup: The defaults which allow the user to generate a plot without modification will be used if users do not put global variables and praphics procedures. For detailed explanations on the global control variables, see the GAUSS manual.
- Calling Graphics Routines: These graphic routines are used to create and display graphics file.

The following is a summary of GAUSS graphics commands.

Four Basic Parts	Example	Comments
Header	*Library pgraph*; *graphset*;	Activate PGRAPH library reset global variables
Data Setup	$x = seqa(0.1, 0.1, 50)$; $y = \sin(x)$;	
Grapics Format	*title*("*Example* XY *graph*");	See Manaual
Graphics Routines	$xy(x, y)$;	See General Design

7.3 General Design

Gauss graphics consists of a set of main graphing procedures and several additional procedures and global variables for customising the output.

Graphic Procedures	Comments
BAR	Bar Graphs
BOX	Box Plots
CONTOUR	Countour Plots
DRAW	Draws graphs using only global variables
HIST	Histogram
HISTP	Percentage histogram
HISTF	Histogram from a vector of frequencies
LOGLOG	Log scaling on both axes
LOGX	Log scaling on X axis
LOGY	Log scaling on Y axis
POLAR	Polar plots
SURFACE	3-D surface with hiddeen line removal
XY	Cartesian graph
XYZ	3-D Cartesian graph

8. ERROR HANDING AND DEBUGGING

As explained above, one of the disadvantages in GAUSS is that GAUSS is too tolerant of mistakes and errors, which are difficult for the computer or user to detect. In this section, we first discuss on the error messages and then some frequently encountered mistakes.

8.1 Error Messages

If there is an error in a programme, then the programme usually terminates and an error message is printed. A single error can lead to the printing of several error messages. In many cases, the first error may be the only real error.

Here is an example. Let x be $T \times N$ matrix and calculate inverse of the variance-covariance matrix of x. If we type

$$\begin{aligned} y &= \det(x * x); \\ & \quad print\ y; \end{aligned}$$

then we have the following error message;

$$c : \backslash lec - gaus \backslash lec - gaus.txt(375) : errorG0036 : Matrices\ are\ not\ conformable$$

The first part of the error message describes the GAUSS file name and the number in brackets represents the number of line where the error occurs. Next, an error code (here 0036) and brief explanation of the error follow. For the detailed information on the error message can be found in the GAUSS manual.

Sometimes error messages are generated by library functions, e.g., MAXLIK, when we use them. In this case, it becomes very difficult to find out what is a real problem in your GAUSS programme. One of the best way may be to check if your GAUSS programme follows all input and output requirement necessary for the library functions.

9. SOME USEFUL TIPS

In this section we consider some aspects of improving the efficiency of programmes.

9.1 Use GAUSS Functions and Matrix

In many cases, there is no significant difference between GAUSS functions and user defined functions mainly due to the development of hardware. However, when we need repeated calculations such as simulations, small changes in a GAUSS code give us significant time saving.

The GAUSS functions are faster than an option written by the user. This is because the maths co-processor has vector processing instructions built into it which the GAUSS standard functions were designed to use fully. A user defined procedure will always have to go through one level of abstraction (writing GAUSS code to be translated into machine instructions). This means that a user programme is unlikely to be more efficient then the GAUSS function, and is probably less.

9.2 Use Comments and Spaces

It is important to put some comments on the right hand side of programme with the comment symbol @. This will help you understand your GAUSS code for a future use of the code. In addition, use space and tab key if possible to represent some parts in your code are sub-codes to the main stream.

9.3 Use Comments Symbol to Detect Errors

In most cases, if a GAUSS code contains tens or hundreds of lines, it becomes very difficult to find where are the errors. In these cases, you can use the comment symbols such as @..@ and $/* ... */$ to check each part where you think errors exist. This is tedious work but effective.

9.4 Use a Simple Data and Compare the Results with Your Expectation

This may be the most powerful test for the final check of your programme. One of the best way is to use the same data as used in other researches and compare your results with those of the other researches. However, in many cases it is difficult to get such data.

Another method is use some artificial numbers which results in predicted results. For example, in CAPM if we use the market portfolio as a dependent variable, then the results we expect is $\alpha = 0$ and $\beta = 1$.

10. MAXIMUM LIKELIHOOD ESTIMATION

Maximum likelihood (ML) in the GAUSS MAXIMUM LIKELIHOOD MODULE is a set of procedures for the estimation of the parameters of models via maximum likelihood method with general constraints on the parameters.

ML solves the general maximum likelihood problem

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log[pdf(x_i; \boldsymbol{\theta})]$$

where N is the number of observations, $pdf(x_i; \boldsymbol{\theta})$ is the probability density function of x_i given a $(k \times 1)$ parameter vector $\boldsymbol{\theta}$. The ML procedure called **MAXLIK** finds values for the parameters in $\boldsymbol{\theta}$ such that the maximum likelihood function value ℓ is maximised.

10.1 Numerical Optimisation

Our question is how to find the value of $\widehat{\boldsymbol{\theta}}$ that maximises $\ell(\boldsymbol{\theta})$. The general approach is to write a procedure that enables a computer to calculate the numerical value of $\ell(\boldsymbol{\theta})$ for any particular values for $\boldsymbol{\theta}$ and the observed data x_i. The approach will be to make a series of different guesses for $\boldsymbol{\theta}$, compare the value of $\ell(\boldsymbol{\theta})$ for each guess and try to infer from these values for $\ell(\boldsymbol{\theta})$ the value $\widehat{\boldsymbol{\theta}}$ for which $\ell(\boldsymbol{\theta})$ is largest. Such methods are described as *numerical maximisation*.

An effective means of solving most maximising problems is by an *iterative algorithm*. That is, beginning from initial values $\boldsymbol{\theta}^{(0)}$, at entry to iteration t, if $\boldsymbol{\theta}^{(t)}$ is not the optimal value for $\boldsymbol{\theta}$, then compute direction vector $\boldsymbol{\Delta}^{(t)}$, step size $\lambda^{(t)}$ (also called line search), and we have

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \lambda^{(t)}\boldsymbol{\Delta}^{(t)}. \tag{10.1}$$

Equation (10.1) suggests that we need two components to numerically maximise the likelihood function; $\lambda^{(t)}$ and $\boldsymbol{\Delta}^{(t)}$. Many theoretical methods have been developed to obtain appropriate sets of $\lambda^{(t)}$ and $\boldsymbol{\Delta}^{(t)}$. See section 5.5, Greene (1997) and 5.7, Hamilton (1994).

In this course, we discuss only one method called *Newton-Raphson*. Consider the above ML function, $\ell(\boldsymbol{\theta})$, and define $\underset{k\times 1}{\mathbf{g}}(\boldsymbol{\theta}^{(0)}) = \frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(0)}}$ and $\underset{k\times k}{\mathbf{H}}(\boldsymbol{\theta}^{(0)}) = -\frac{\partial^2 \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}\partial \boldsymbol{\theta}'}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(0)}}$, where $\boldsymbol{\theta}^{(0)}$ denotes an initial guess as to the value of $\boldsymbol{\theta}$. Consider approximating $\ell(\boldsymbol{\theta})$ with a second-order Taylor series around $\boldsymbol{\theta}^{(0)}$. Then we have

$$\ell(\boldsymbol{\theta}) \cong \ell(\boldsymbol{\theta}^{(0)}) + [\mathbf{g}(\boldsymbol{\theta}^{(0)})]'[\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}] - \frac{1}{2}[\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}]'[\mathbf{H}(\boldsymbol{\theta}^{(0)})][\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}]. \tag{10.2}$$

The idea is to choose $\boldsymbol{\theta}$ to maximise the above equation. The necessary and sufficient conditions for the maximisation problem is that the first derivative of the equation against $\boldsymbol{\theta}$ equals to zero

$$\frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cong [\mathbf{g}(\boldsymbol{\theta}^{(0)})] - [\mathbf{H}(\boldsymbol{\theta}^{(0)})][\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}] = 0 \tag{10.3}$$

and the second derivative should be negative, $\frac{\partial^2 \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}'} \cong -[\mathbf{H}(\boldsymbol{\theta}^{(0)})] < 0$, requiring positive definiteness of $[\mathbf{H}(\boldsymbol{\theta}^{(0)})]$. When we re-arrange equation (10.3), we have

$$\boldsymbol{\theta} = \boldsymbol{\theta}^{(0)} + [\mathbf{H}(\boldsymbol{\theta}^{(0)})]^{-1}[\mathbf{g}(\boldsymbol{\theta}^{(0)})]. \tag{10.4}$$

Equation (10.4) suggests an improved estimate of $\boldsymbol{\theta}$ can be obtained.. This can be rewritten as

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + [\mathbf{H}(\boldsymbol{\theta}^{(t)})]^{-1}[\mathbf{g}(\boldsymbol{\theta}^{(t)})]. \tag{10.5}$$

Note that the Newton-Raphson method only suggests the search direction, $\boldsymbol{\Delta}^{(t)} = [\mathbf{H}(\boldsymbol{\theta}^{(t)})]^{-1}[\mathbf{g}(\boldsymbol{\theta}^{(t)})]$. Therefore, the value of $\ell(\boldsymbol{\theta})$ is calculated in that direction at several points, and the best value determines the length of the step. Therefore we have

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \lambda^{(t)}[\mathbf{H}(\boldsymbol{\theta}^{(t)})]^{-1}[\mathbf{g}(\boldsymbol{\theta}^{(t)})]$$

where $\lambda^{(t)}$ is a scalar controlling the step length.

10.2 Maximum Likelihood Module

To use the procedures in the GAUSS MAXIMUM LIKELIHOOD MODULE (the explanation below is based on MAXLIK version 4), we have to make the MAXLIK library active. The MAXLIK library is activated by including maxlik in the library statement at the top of your programme;

library maxlik;
#include maxlik.ext;

The second line will make any right hand references to the global variables.

Finally, with the above statements, we can include the following instruction in order to reset global variables in succeeding executions for the command file;

maxset;

Now we are ready to use the MAXLIK module.

The format of the MAXLIK is

$$\{est, ML\_value, grad, cov, retcode\} = MAXLIK(dataset, vars, \&fct, start);$$

10.2.1 Input

- *dataset (***x***)*: string containing name of GAUSS data set, or name of data matrix stored in memory.
- *vars*: character vector of labels selected for analysis, or numeric vector of column numbers in data set of variables selected for analysis.
- *fct*: the name of a procedure that returns either the log-likelihood for one observation or a vector of log-likelihoods for a matrix of observations.
- *start* ($\boldsymbol{\theta}^{(0)}$)- a $K \times 1$ vector of start values.

10.2.2 Output

- $est(\widehat{\boldsymbol{\theta}})$: $K \times 1$ vector, estimated parameters.
- $ML\_\ value(\ell(\widehat{\boldsymbol{\theta}}))$: scalar, function at maximum (mean log-likelihood).
- $grad(\mathbf{g}(\widehat{\boldsymbol{\theta}}))$: $K \times 1$ vector, gradient evaluated at x.
- *cov*: $K \times K$ matrix, covariance matrix of the parameters.
- *retcode*: scalar, return code:
  - 0 normal convergence
  - 1 forced exit
  - 2 maximum number of iterations exceeded
  - 3 function calculation failed
  - 4 gradient calculation failed
  - 5 Hessian calculation failed
  - 6 step length calculation failed
  - 10 secant update failed
  - 11 maximum time exceeded
  - 20 Hessian failed to invert
  - 34 data set could not be opened
  - 99 termination condition unknown

10.2.3 Procedure For Computing Likelihood Function

The user must provide a procedure (fct) for computing the log-likelihood for a matrix of observations. The procedure must have two input arguments; a vector of parameter values and the data matrix. The output argument is the log-likelihood for the observations in the second argument evaluated at the parameters values in the first argument.

$$proc\ fct(start, dataset)$$

INPUT

- *start* ($\boldsymbol{\theta}^{(0)}$): vector of parameters of model
- *dataset* ($\mathbf{x}$): one or more rows of the data set

OUTPUT

- The output is the value of the log-likelihood evaluated at the parameters for the dataset. if you write the procedure so that a vector of log-likelihoods is returned, then set __row=0 (this is default). Otherwise you have to set __row=1.

10.2.4 Global Variables

It is important to understand how we maximise likelihood functions. The MAXLIK module in GAUSS allows us to choose the method we are going to use in maximising likelihood functions.

As discussed above, there are several methods to calculate direction vector $\mathbf{\Delta}^{(t)}$ and line search $\lambda^{(t)}$. In addition, there are many options in the GAUSS MAXLIK module to avoid possible problems we may encounter during maximising a likelihood function. We do not discuss all of them in this course. For detailed discussions, please refer to the MAXLIK manual.

Optimization Method

$$\_max\_Algorithm = 2;$$

- 1, SD (steepest descent)
- 2, BFGS (Broyden, Fletcher, Goldfarb, Shanno) (default)
- 3, DFP (Davidon, Fletcher, Powell)
- 4, NEWTON (Newton-Raphson) (explained above)
- 5, BHHH
- 6, Polak-Ribiere Conjugate Gradient

Line Search

$$\_max\_LineSearch = 2;$$

- 1, steplength = 1
- 2, STEPBT (default)
- 3, HALF
- 4, BRENT
- 5, BHHHSTEP

In some line search methods such as BRENT, we need to specify maximum number of tries. Note that in these cases, the MAXLIK uses a specified number of tries (default value is 100) to get the optimal value of line search, which needs a lot of calculation time. We can control maximum number of tries in line search by using the following statement.

$$\_max\_MaxTry = 100;$$

Convergence Tolerance

Since the MAXLIK is a module for numerical optimisation (it is not analytical maximisation), we have to specify when to stop optimisation. If $\lambda^{(t)}\boldsymbol{\Delta}^{(t)} = 0$ and thus $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)}$ in equation (10.1) and the likelihood function is globally concave, we can get ML estimates which maximises the likelihood function.

The GAUSS MAXLIK needs all elements of the gradient vector to be zero, $\mathbf{g}(\boldsymbol{\theta}^{(t)}) = \mathbf{0}$ to stop maximisation and report the results. Since the elements of the gradient vector are calculated numerically, the question is how close the gradient vector should be in order to conclude that we have the ML estimates. We can specify the value using the following global variable.

$$\_max\_GradTol = 1e-5;$$

This is a default value and is enough for most financial researches. When we use smaller values such as $1e-6$, the calculation time increases dramatically.

Covariance Matrix of Parameters

The following global variable specifies which method will be used to calculate covariance matrix of estimated parameters.

$$\_max\_CovPar = 0;$$

- 0, the inverse of the final information matrix from the optimization is returned (default).
- 1, the inverse of the second derivatives is returned.
- 2, the inverse of the cross-product of the first derivatives is returned.
- 3, the heteroskedastic-consistent covariance matrix is returned.

Maximum Iterations

In some cases, numerical optimisations are tried infinitely which need a lot of calculation time. In these cases we may suspect that there are some problems in our GAUSS code or the starting values we provided. There are two methods to prevent this.

The first method is to control the number of iterations. For example,

$$\_max\_MaxIters = 1000;$$

The other method is to specify the calculation time in minutes..

$$\_max\_MaxTime = 1e+5;$$

This amounts to about 10 weeks.

Maximum Radius

If this value is set to a nonzero value (1e-2, say) and all other line search methods fail, then MAXLIK will attempt a random direction with radius determined by _max_RandRadius. This global variable is useful when some constraints are introduced to the parameters we are going to estimate.

$$\_max\_RandRadius = 1e-2;$$

Activation and Deactivation of Parameters

This global variable allows us to fix a parameter to its starting value. This is useful when we wish to try different models with different sets of parameters without having to re-edit the function.

$$\_max\_Active = \{1, 1, 0\};$$

This global variable must be a vector of the same length as the starting vector. Set elements to one for an active parameter and to zero for a fixed one.

Other Global Variables

Output This determines printing of intermediate results. Generally when this variable is nonzero and there are some kind of printing during the iterations, the time of the iterations is degraded.

$$\_\_output = 1;$$

- 0, nothing is written
- 1, serial ASCII output format suitable for disk files of printers
- 2, output is suitable for screen only.

In the MAXLIK, we have a procedure called MAXPRT to print the final results of the ML estimation. We can use

$$\{est, ML\_value, grad, cov, retcode\} = MAXPrt(MAXLIK(dataset, vars, \&fct, start));$$

or

$$print\ MAXPrt(est, ML\_value, grad, cov, retcode);$$

Title Title of run can be in the form of

$$\_\_title = "GARCH(1,1)\ Estimates\ for\ British\ Petroleum\ Daily\ Volatility";$$

10.3 An Example - GARCH(1,1) Estimation

Let us discuss how to estimate parameters from a maximum likelihood function with an example, GARCH(1,1). A GARCH(1,1) process introduced by Bollerslev (1986) is

$$\begin{aligned} u_t &= \sigma_t \sqrt{h_t} \\ h_t &= \alpha_0 + \alpha_1 h_t + \alpha_2 u_t^2 \end{aligned}$$

where u_t is a residual process, $\sigma_t \tilde{} N(0,1)$. Note that $E(u_t) = E(\sigma_t)E(\sqrt{h_t}) = 0$, and $E(u_t^2) = E(\sigma_t^2)E(h_t) = E(h_t)$.

$library\ maxlik;$	Activates maxlik library.
$\#include\ maxlik.ext;$	Makes any right hand references to the global variables.
$fname = "C:\\lec-gaus\\Bp-D.xls";$	Reads $bp-d.xls$ file.
$range = "c8:c1462";$	Reads range specified.
$\{bp, names\} = import(fname, range, 1);$	Variable name is 'bp'.
$var = vcx(bp);$	Calculates variance of bp.
$u = (bp - meanc(bp))\hat{}2;$	Calculates daily volatility series.
$maxset;$	Resets global variables.
$\_max\_active = \{1, 1, 1\};$	Activates all three parameters.
$\_max\_gradtol = 1e-5;$	Sets convergence criterion for gradient.
$\_max\_algorithm = 5;$	Selects algorithm (BHHH).
$\_max\_linesearch = 5;$	Selects line search method (BHHH).
$\_max\_maxiters = 1000;$	Maximum number of iterations
$\_max\_maxtry = 5;$	Maximum number of tries in line search
$\_max\_randradious = 1e-3;$	Sets radius for a random direction.
$\_max\_covpar = 3;$	Covariance matrix type (parameters).
$\_\_output = 1;$	Printing option of intermediate retults
$iv = 0.001\|0.97\|0.02;$	Vector of Starting values
$\{est, fmax, grad, cov, retcode\} =$ $maxlik(u, 1, \&lr, iv);$	Call Maximum Likelihood Mudule
$proc\ lr(miv, u);$	Procedure for GARCH(1,1) process
$LOCAL\ a0, a1, a2, h, t, ll;$	Sets local variables.
$a0 = miv[1]; a1 = miv[2]; a2 = miv[3];$	Assigns starting values to parameters.
$if\ a0. <= 0; ll = -1000000; goto\ skip; endif;$	Conditions on the parameters
$if\ a1. < 0; ll = -1000000; goto\ skip; endif;$	"
$if\ a2. < 0; ll = -1000000; goto\ skip; endif;$	"
$if\ (a1 + a2). > 1; ll = -1000000;$ $gotoskip; endif;$	„
$h = zeros(rows(u), 1);$	Demensions of conditional volatility.
$h[1] = a0 + (a1 + a2). * var$	
$t = 2;$	
$do\ until\ t. > rows(u);$	
$h[t] = a0 + a1. * h[t-1] + a2. * u[t-1];$	GARCH(1,1) process
$t = t + 1;$	
$endo;$	
$ll = -0.5. * ln(2. * pi)$ $-0.5. * ln(h[2 : rows(u)])$ $-0.5. * (u[2 : rows(u)]./h[2 : rows(u)]);$	Log-likelihood function
$skip:$	
$RETP(ll);$	Returns function values(vector).
$endp;$	End of procedure
$\_\_title = "GARCH(1,1)\ Estimates";$	Title name
$print\ maxprt(est, fmax, grad, cov, retcode);$	Prints the results.

11. COMMANDS BY CATEGORY

11.1 Mathematical Functions

11.1.1 Scientific Functions

Commands	Stetements	Explanations
ABS	$y = ABS(x);$	Returns absolute value of argument.
BESSELJ	$y = BESSELJ(n, x);$	Computes Bessel function, first kind.
BESSELY	$y = BESSELY(n, x);$	Computes Bessel function, second kind.
COS	$y = COS(x);$	Computes cosine.
EXP	$y = EXP(x);$	Computes the exponential function of z.
GAMMA	$y = GAMMA(x);$	Computes gamma function value.
LN	$y = LN(x);$	Computes the natural log of each element.
LOG	$y = LOG(x);$	Computes the $\log_{10}$ of each element.
PI	$y = PI;$	Returns π.
SIN	$y = SIN(x);$	Computes sine.
SQRT	$y = SQRT(x);$	Computes the square root of each element.

11.1.2 Differentiation and Integration

Commands	Stetements	Explanations
GRADP	$y = GRADP(\&f, x0);$	Computes first derivative of a function.
HESSP	$y = HESSP(\&f, x0);$	Computes second derivative of a function.
INTGRAT2	$y = INTGRAT2(\&f, xl, gl);$	Integrate a 2-dimensional function over a user-defined region.
INTGRAT3	$y = INTGRAT3(\&f, xl, gl, hl);$	Integrate a 3-dimensional function over a user-defined region.
INTQUAD1	$y = INTQUAD1(\&f, xl);$	Integrate a 1-dimensional function.
INTQUAD2	$y = INTQUAD2(\&f, xl, yl);$	Integrate a 2-dimensional function over a user-defined rectangular region.
INTQUAD3	$y = INTQUAD3(\&f, xl, yl, zl);$	Integrate a 3-dimensional function over a user-defined rectangular region.
INTSIMP	$y = INTSIMP(\&f, xl, tol);$	Integrate by Simpson's method.

- GRADP and HESSP use a finite difference approximation to compute the first and second derivatives. Use GRADP to calculate a Jacobian.
- INTQUAD1, INTQUAD2, and INTQUAD3 use Gaussian quadrature to calculate the integral of the user-defined function over a rectangular region.
- To calculate an integral over a region defined by functions of X and y, use INTGRAT2 and INTGRAT3.
- To get a greater degree of accuracy than that provided by INTQUAD1, use INTSIMP for one-dimensional integration.

11.1.3 Linear Algebra

Commands	Stetements	Explanations
CHOL	$y = chol(x);$	Computes Cholesky decomposition.
CHOLDN	$y = chol(C, x);$	Performs Cholesky downdate on an upper triangular matrix.
CHOLSOL	$y = chol(b, C);$	Solves a system of equations given the Cholesky factorization of a matrix.
CHOLUP	$y = chol(C, x);$	Performs Cholesky update on an upper triangular matrix.
DET	$y - DET(x);$	Computes determinant of square matrix.
DETL	$y = DETL;$	Computes determinant of decomposed matrix.
INV	$y = INV(x);$	Inverts a matrix.
INVPD	$y = INVPD(x);$	Inverts a positive definite matrix.
RANK	$y = RANK(x);$	Computes rank of a matrix.

- INV, INVPD and the "/" operator can all be used to solve linear systems of equations.

11.1.4 Eigenvalues

Commands	Statements	Explanations
EIG	$va = EIG(x);$	Computes eigenvalues of general matrix.
EIGV	$\{va, ve\} = EIGV(x);$	Computes eigenvalues and eigenvectors of general matrix.

11.1.5 Random Numbers

Commands	Stetements	Explanations
RNDN	$y = RNDN(r, c);$	Creates a matrix of normally distributed random numbers.
RNDNS	$y = RNDNS(r, c, s);$	Creates a matrix of normally distributed random numbers (specify seed).
RNDSEED	$RNDSEED\ seed;$	Changes seed of random number generator.
RNDU	$y = RNDU(r, c);$	Creates a matrix of uniformly distributed random numbers.
RNDUS	$y = RNDUS(r, c, s);$	Creates a matrix of uniformly distributed random numbers (specify seed).

11.1.6 Statistical Functions

Commands		Explanations
CORRM	$cx = CORRM(m);$	Computes correlation matrix of a moment matrix.
CORRVC	$cx = CORRVC(vc);$	Computes correlation matrix from a variance-covariance matrix.
CORRX	$cx = CORRX(x);$	Computes correlation matrix.
CROSSPRD	$z = CROSSPRD(x, y);$	Computes cross product.
MEANC	$y = MEANC(x);$	Computes mean value of each column of a matrix.
MEDIAN	$y = MEDIAN(x);$	Computes medians of the columns of a matrix.
MOMENT	$y = MOMENT(x, d);$	Computes moment matrix ($x'x$) with special handling of missing values.
MOMENTD	$y = MOMENTD(dataset, vars);$	Computes moment matrix from a dataset.
OLS	$\{vnam, m, b, std, vc, stderr, sigma, cx, rsq, resid, dwstat\} = OLS(dataset, depvar, indvars);$	Computes least squares regression of dataset.
STDC	$y = STDC(x);$	Computes standard deviation of the columns of a matrix.
TOEPLITZ	$y = TOEPLITZ(x);$	Computes Toeplitz matrix from column vector.
VCM	$y = VCM(m);$	Computes a variance-covariance matrix from a moment matrix.
VCX	$y = VCX(x);$	Computes a variance-covariance matrix from a data matrix.

11.1.7 Statistical Distributions

Commands	Stetements	Explanations
CDFBVN	$c = CDFBVN(h, k, r);$	Computes lower tail of bivariate normal CDF.
CDFCHIC	$y = CDFCHIC(x, n);$	Computes complement of CDF of χ^2.
CDFCHINC	$y = CDFCHINC(x, v, d);$	Computes integral of noncentral χ^2.
CDFFC	$y = CDFFC(x, n1, n2);$	Computes complement of CDF of F.
CDFFNC	$y = CDFFNC(x, v1, v2, d);$	Computes integral of noncentral F.
CDFN	$n = CDFN(x);$	Computes integral of normal distribution: lower tail, or CDF.
CDFNC	$nc = CDFNC(x);$	Computes complement(1-CDF) of normal distribution (uppertail).
CDFTNC	$y = CDFTNC(x, v, d);$	Computes integral of noncentral t-distribution.
PDFN	$y = PDFN(x);$	Computes standard normal probability density function.

11.1.8 Series and Sequence Functions

Commands	Stetements	Explanations
SEQA	$y = SEQA(start, inc, n);$	Creates an additive sequence.
SEQM	$y = SEQM(start, inc, n);$	Creates a multiplicative sequence.

11.1.9 Precision Control

Commands	Stetements	Explanations
ROUND	$y = ROUND(x);$	Round to the nearest integer.
TRUNC	$y = TRUNC(x);$	Truncate toward 0.

11.2 Matrix Manipulation

11.2.1 Creating Vectors and Matrices

Commands	Stetements	Explanations
EYE	$y = EYE(n);$	Creates identity matrix.
LET	$LET\ x = constant\_list;$	Creates matrix from list of constants.
ONES	$y = ONES(r, c);$	Creates a matrix of ones.
ZEROS	$y = ZEROS(r, c);$	Creates a matrix of zeros.

11.2.2 Size, Ranking, and Range

Commands	Stetements	Explanations
COLS	$y = COLS(x);$	Returns number of columns in a matrix.
COUNTS	$y = COUNTS(x, v);$	Returns number of elements of a vector fallingin specified ranges.
INDEXCAT	$y = INDEXCAT(x, v);$	Returns indices of elements falling within a specified range.
MAXC	$y = MAXC(x);$	Returns largest element in each column of a matrix.
MAXINDC	$y = MAXINDC(x);$	Returns row number of largest element in each column of a matrix.
MINC	$y = MINC(x);$	Returns smallest element in each column of a matrix.
MININDC	$y = MININDC(x);$	Returns row number of smallest element in each column of a matrix.
ROWS	$y = ROWS(x);$	Returns number of rows in a matrix.
CUMPRODC	$y = CUMPRODC(x);$	Computes cumulative products of each column of a matrix.
CUMSUMC	$y = CUMSUMC(x);$	Computes cumulative sums of each column of a matrix.
PRODC	$y = PRODC(x);$	Computes the product of each column of a matrix.
SUMC	$y = SUMC(x);$	Computes the sum of each column of a matrix.

11.2.3 Miscellaneous Matrix Manipulation

Commands	Stetements	Explanations
REV	$y = REV(x);$	Reverses the order of rows of a matrix.
SHIFTR	$y = SHIFTR(x, s, f);$	Shifts rows of a matrix, filling in holes with a specified value.
RESHAPE	$y = RESHAPE(x, r, c);$	Reshapes a matrix to new dimensions.
VEC	$yc = VEC(x);$	Stacks columns of a matrix to form a single column.
VECH	$v = VECH(x);$	Reshapes the lower triangular portion of a symmetric matrix into a column vector.
XPND	$x = XPND(v);$	Expands a column vector into a symmetric matrix.
DELIF	$y = DELIF(x, e);$	Deletes rows from a matrix using a logical expression.
DIAG	$y = DIAG(x);$	Extracts the diagonal of a matrix.
SELIF	$y = SELIF(x, e);$	Selects rows from a matrix using a logical expression.

11.3 Data Handling

11.3.1 Sorting

Commands	Stetements	Explanations
SORTC	$y = SORTC(x, c);$	Quick-sort rows of matrix based on numeric key.
SORTCC	$y = SORTCC(x, c);$	Quick-sort rows of matrix based on character key.
SORTD	$SORTD(infile, outfile, keyvar, keytyp);$	Sort data set on a key column.
SORTHC	$y = SORTHC(x, c);$	Heap-sort rows of matrix based on numeric key.
SORTHCC	$y = SORTHCC(x, c);$	Heap-sort rows of matrix based on character key.
SORTIND	$y = SORTIND(x);$	Returns a sorted index of a numeric vector.
SORTINDC	$y = SORTINDC(x);$	Returns a sorted index of a character vector.
SORTMC	$y = SORTMC(x, v);$	Sort rows of matrix on the basis of multiple columns.

11.4 Programme Control

11.4.1 Branching

Commands	Explanations
IF..ENDIF	Conditional branching
GOTO	Unconditional branching
POP	Retrieve GOTO arguments

11.4.2 Looping

Commands	Explanations
BREAK	Jump out the bottom of a DO loop.
CONTINUE	Jump to the top of a DO loop.
DO WHILE..ENDO	Loop if TRUE.
DO UNTIL..ENDO	Loop if FALSE.

11.4.3 Procedures

Commands	Explanations
LOCAL	Declare variables local to a procedure.
PROC	Begin definition of multi-line procedure.
RETP	Return from a procedure.
ENDP	Terminate a procedure definition.

11.4.4 Libraries

Commands	Explanations
CALL	Call function and discard return values.
DECLARE	Initialize variables at compile time.
EXTERNAL	External symbol definitions.
LIBRARY	Set up list of active libraries.
LIB	Build or update a GAUSS library.

11.5 Time and Date Functions

Commands	Explanations
DATE	Returns current system date.
DATESTR	Formats date as "mm/dd/yy".
DAYINYR	Returns day number of a date.
ETDAYS	Difference between two times in days.
ETHSEC	Difference between two times in 100ths of a second.
ETSTR	Convert elapsed time to string.
HSEC	Returns elapsed time since midnight in 100ths of a second.
TIME	Returns current system time.
TIMESTR	Format time as "hh:mm:ss".

11.6 Output Functions

Commands	Explanations
OUTPUT	Redirect PIUNT statements to auxiliary output.
OUTWIDTH	Set line width of auxiliary output.
PRINT	Print to screen.
SCREEN ON	Direct PRINT statements to screen.
SCREEN OFF	Suppress PRINT statements to screen.
FORMAT	Defines format of matrix printing.

References

GAUSS; A Beginner'ss Guide, F. Ritchie,

(Web site: http://scottie.stir.ac.uk/~fri01/gauss/gauss.html)

GAUSS Programming for Econometricians - a web-based course,

(Web site: http://eclab.econ.pdx.edu/gpe/toc.htm)

GAUSS; Volume 1, System and Graphics Manual, Aptech Systems, Ltd.

GAUSS; Command reference Manual, Aptech Systems, Ltd.

Greene, W. H., Econometric Analysis, 3rd ed., 1997, Prentice Hall.

Hamilton, J. D., Time Series Analysis, 1994, Princeton.