Using Gauss for Econometrics

Carter Hill and Lee Adkins

August 30, 2001

Contents

Chapter 1

Introduction

This manual is to be used by econometrics students who are trying to do their homework using **GAUSS**. Much of the introductory material is taken either from the **GAUSS** documentation or from **GAUSS**'s on-line help. The purpose of the introductory material is to get you started using **GAUSS** and to make it easier to use the much-improved on-line help system. It is not intended to replace the **GAUSS** documentation, but we realize that such documentation is not always readily available even when the **GAUSS** site you are using is properly licence's (like ours at OSU).
The second portion of the manual consists of chapters written by Carter Hill and formerly published by John Wiley and Sons under the title, *Learning Econometrics Using **Gauss***. The book once sold as a supplement to Judge, et al. *Introduction to the Theory and Practice of Econometrics*. Wiley (mistakenly, we believe) allowed *Learning Econometrics* to go out of print and has no intention of reviving the product at this time.
The combination of *Learning Econometrics Using **GAUSS*** and the introductory material on using **GAUSS** that you are holding in your hands has been rapidly thrown together; the project is in its preliminary stages and this manual is being offered 'as is' with the hope that further refinements will make it a usable supplement to econometrics courses.

1.1 Using the On-line Help

The focus of the first part of this manual is to enable you to get started using **GAUSS**. The **GAUSS** commands are effectively documented in the on-line help system. From what we can tell, the documentation from the on-line help is nearly identical to that in the printed documentation. Hence, one of the things you need to learn is how to use the on-line help to get the information you need. **GAUSS**'s on-line HELP system can provide help on all **GAUSS** functions and operators, and all user-defined functions. It can also be used to display text files.

To get into the Help system, press **Alt-H**. The following screen will be shown:

```
TO GET:                                                    TYPE
------------------------------------------------------------------
Help on HELP (this screen) ..................... HELP
A Listing of Categories ........................ @CATS
A Listing of GAUSS Intrinsic Functions ......... @CMDS
A Table of GAUSS Operators ..................... @OPERS
A Table of Command Mode Keys ................... @CKEYS
A Table of Edit Mode Keys ...................... @EKEYS

============================================================
Help PgDn PgUp Home Esc
```

From here, type **H** to get the `"Help on:"` prompt:

```
============================================================
Help on:
```

Now type the name desired at the prompt. For instance, to get a list of **GAUSS** intrinsic functions type `@cmds`, i.e.,

```
============================================================
Help on:  @cmds
```

You can also use the '*' and '?' wildcards when entering requests; '*' stands for 0 or more instances of any character, and '?' stands for 1 instance of any character. **GAUSS** will display a window of all items that match your request, from which you can make selections.
If you specify a path when asking for help on a file, **GAUSS** will look only in the directory specified. If you do not specify a path, **GAUSS** will search for the file first in the current directory, then along the source path. If the file has a .LCG extension, **GAUSS** will search for it in the library path. This makes it easy to get help on source code and library files. Note, you cannot use wildcards in filenames.
After exiting HELP, you can press **F9** from COMMAND or EDIT mode to return to the last HELP screen you viewed.

1.1.1 Functions, Operators, and Categories

GAUSS refers to **GAUSS**'s canned functions and operators as *intrinsics*. In using the on-line help on intrinsics and categories the following keystroke commands are available:

H(elp), **Alt-H**	Request help on new subject
Pg Dn	Page down through intrinsics/categories
Pg Up	Page up through intrinsics/categories
Home	Go to first HELP screen ("About Help")
Esc	Exit HELP

1.1.2 Run-Time Library and User-Defined Functions

The following commands are available when viewing help on Run-Time Library or user-defined functions, or displaying a text file:

H(elp),**Alt-H**	Request help on new subject
G(oto),**Alt-G**	Go to line number
S(earch),**F5**	Search for text
Pg Dn	Page down through file
Pg Up	Page up through file
Up,**Down**	Scroll up/down in file
Left,**Right**	Pan left/right in file
Home	Go to top of file
End	Go to end of file
Esc	Exit HELP

The following commands are also available, although they are not on the menu:

Alt-S,**Shift-F5**	Search again, same text
Ctrl-S,**Ctrl-F5**	Toggle text search case sensitivity

1.1.3 Item Selection

When you use wildcards in a request, **GAUSS** compiles a list of all items that match and displays them in a window. The following commands are available when selecting an item from the window:

Tab	Request help on new subject
Pg Dn	Page down through list
Pg Up	Page up through list
Up,**Down**	Scroll up/down in list
Home	Go to top of list
End	Go to end of list
chars...	Construct speed search pattern
Backspace	Delete last speed search char
Enter	Select highlighted item
Esc	Exit HELP

You can do a speed search through the list by typing the beginning characters of an item. The highlight will move to the first item that matches what you type. (This is why **Tab** is used instead of 'H' to get the "Help on:" prompt.) Valid characters are added to the search pattern; characters that don't match are ignored. **Backspace** deletes the last character in the pattern. Moving the highlight up or down with the cursor keys deletes the entire pattern.

1.1.4 Entering Requests

The following commands are available when entering Help, Goto, and Search requests:

Home	Go to beginning of entry
End	Go to end of entry
Left,**Right**	Move cursor left/right within entry
Backspace	Delete character to left of cursor
Del	Delete character under cursor
Up,**Down**	Scroll up/down through previous entries
Enter	Accept entry
Esc	Abort entry, return to HELP

Note that **GAUSS** remembers previous requests. You can scroll up and down through them and modify or reuse old entries.
Below you will find a summary of the categories that on-line help covers. To obtain help on a particular topic, type the expression found on the right-hand-side of the table.

```
TO GET HELP ON:                                                    TYPE

HELP..................................................... HELP

BASIC MATHEMATICAL/SCIENTIFIC OPERATIONS
    Rounding, Truncating and Precision Control ................. @PRC
    Mathematical and Scientific Functions ...................... @MATH
    Differentiation and Integration Routines ................... @DI
    Root Finding, Polynomial Multiplication and Interpolation .. @POLY
    Frequency Transforms - FFT's ............................... @FREQ
    Random Number Generators and Seeds ......................... @RND
    Complex Number Operations .................................. @COMPLEX
    Fuzzy Comparison Operators ................................. @FUZZY
DATA TRANSFORMATIONS ..................................... @DATALOOP

STATISTICS
    Cumulative Distribution Functions .......................... @CDF
    Descriptive Statistics ..................................... @STAT
    Linear Regression .......................................... @REG

MATRIX CREATION AND MANIPULATION
    Creating Matrices .......................................... @CREATE
    Rank and Size of Matrices .................................. @SIZE
    Submatrix Extraction ....................................... @XTRCT
    Basic Row, Column and Set Operations ....................... @RC
    Matrix Element Manipulation ................................ @MAN

DATA HANDLING    (see also DATA TRANSFORMATIONS above)
    Working with Data Sets ..................................... @DATA
    Working with Variables in Data Sets ........................ @VAR
    Loading, Saving, and Editing Matrices, Procs, etc. ......... @LSE
    Coding Data Matrices and Vectors ........................... @CODE
    Working with Missing Values ................................ @MISS
    Sorting Routines ........................................... @SORT
```

```
LINEAR ALGEBRA
    Eigenvalue and Eigenvector Routines ..... @EIG
    Matrix Decompositions ..... @DEC
    Matrix Inversions and Linear System Solutions ..... @INV

PROGRAMMING STATEMENTS
    Program Execution Control - RUN, STOP, etc. ..... @EXE
    Branching, Conditional and Unconditional ..... @BRANCH
    Looping Control ..... @LOOP
    Writing Subroutines ..... @SUB
    Compiler Directives ..... @DIRECTIVES

EXTENDING \Gauss
    Writing Procedures ..... @PROCEDURE
    Using External Functions ..... @EXT
    Using the Library System ..... @LIB
    Using the Foreign Language Interface ..... @FLI

PUBLICATION QUALITY GRAPHICS ..... @PQG

OTHER
    DOS Shell Commands ..... @SHELL
    Workspace Management ..... @SPACE
    Error Handling and Debugging ..... @ERR
    String Handling Routines ..... @STR
    String Arrays ..... @STRARRAY
    Time and Date Functions ..... @TIME
    General File I/O ..... @FILEIO
    Console Operations ..... @CON
    Printer Operations ..... @PRTR
    Printing, Plotting and Screen Output ..... @DISPLAY
    Compiling Programs ..... @COMPILE
```

Below is a list of intrinsic commands available in **GAUSS** . Specific information on how to use these commands is available using the on-line help. For instance,

```
Help on: abs
```

yields the following information

```
                                        ABS
Purpose:    Returns the absolute value of its argument.

Format:     y = ABS(x);

Input:      x     NxK matrix.

Output:     y     NxK matrix containing the absolute values of x.
```

The other commands available are:

ABS	CDFNC	COMPILE	DET	ENDP	FILES
ATAN	CDFNI	COMPLEX	DETL	ENVGET	FLOOR
ATAN2	CDFTC	CON	DFREE	EOF	FMOD
BALANCE	CDFTCI	CONJ	DIAG	ERF	FN
BAND	CDFTVN	CONS	DIAGRV	ERFC	FOPEN
BANDCHOL	CDIR	CONTINUE	DISABLE	ERROR	FORMAT
BANDCHOLSOL	CEIL	CONV	DO	ERRORLOG	FPUTS
BANDLTSOL	CHOL	CORELEFT	DOS	EXEC	FPUTST
BANDRV	CHOLDN	COS	ED	EXP	FSEEK
BANDSOLPD	CHOLSOL	COUNTS	EDIT	EXTERNAL	FSTRERROR
BESSELJ	CHOLUP	CREATE	EDITM	EYE	FTELL
BESSELY	CHRS	CROUT	EIG	FCHECKERR	FTOCV
BREAK	CLEAR	CROUTP	EIGH	FCLEARERR	FTOS
CALL	CLEARG	CSRCOL	EIGHV	FFLUSH	GAMMA
CALLEXE	CLOSE	CSRLIN	EIGV	FFT	GETF
CDFBETA	CLOSEALL	CSRTYPE	ELSE	FFTI	GETNAME
CDFBVN	CLS	CVTOS	ELSEIF	FFTN	GOSUB
CDFCHIC	COLOR	DATE	ENABLE	FGETS	GOTO
CDFFC	COLS	DEBUG	END	FGETSA	GRAPH
CDFGAM	COLSF	DECLARE	ENDIF	FGETSAT	HASIMAG
CDFN	COMLOG	DELETE	ENDO	FGETST	HESS
HSEC	LOADEXE	MINC	PRINT	RFFTP	SEEKR
IF	LOADF	MININDC	PRINTDOS	RNDCON	SEQA
IMAG	LOADK	MISS	PRINTFM	RNDMOD	SEQM
INDCV	LOADM	MISSRV	PROC	RNDMULT	SETVMODE
INDNV	LOADP	MOMENT	PRODC	RNDN	SHIFTR
INT	LOADS	MSYM	PUSH	RNDNS	SHOW
INV	LOCAL	NDPCHK	RANKINDX	RNDSEED	SIN
INVPD	LOCATE	NDPCLEX	RCONDL	RNDU	SLEEP
INVSWP	LOG	NDPCNTRL	READR	RNDUS	SOLPD
ISCPLX	LOWER	NEW	REAL	ROTATER	SORTC
ISCPLXF	LPOS	ONES	RECSERAR	ROUND	SORTCC
ISMISS	LPRINT	OPEN	RECSERCP	ROWS	SORTHC
KEY	LPWIDTH	OUTPUT	RESHAPE	ROWSF	SORTHCC
KEYW	LSHOW	OUTWIDTH	RETP	RUN	SORTIND
KEYWORD	LTRISOL	PACKR	RETURN	SAVE	SORTINDC
LET	LU	PDFN	REV	SAVEALL	SQRT
LIB	LUSOL	PI	RFFT	SCALERR	STDC
LIBRARY	MATRIX	PLOT	RFFTI	SCALMISS	STOCV
LINE	MAXC	PLOTSYM	RFFTIP	SCHUR	STOF
LN	MAXINDC	POP	RFFTN	SCREEN	STOP
LOAD	MEANC	PRCSN	RFFTNP	SCROLL	STRINDX
STRING	TYPECV				
STRLEN	TYPEF				
STRRINDX	UNION				
STRSECT	UNIQINDX				
SUBMAT	UNIQUE				
SUMC	UNTIL				
SVDCUSV	UPPER				
SVDS	USE				

=	%	==	EQ	$==	AND
+	[	.==	.EQ	.$==	.AND
$+	]	>=	GE	$>=	EQV

```
SVDUSV          UTRISOL          -       :     .>=    .GE    .$>=    .EQV
SYSSTATE        VALS             *       ,      >      GT     $>      NOT
SYSTEM          VARGET          .*       .     .>     .GT    .$>     .NOT
TAB             VARGETL          /       |      <=     LE     $<=     OR
TAN             VARPUT          ./      $|     .<=    .LE    .$<=    .OR
TEMPNAME        VARPUTL          *~      ~      <      LT     $<      XOR
TIME            VEC             .*.     $~     .<     .LT    .$<     .XOR
TRACE           VECH             ^       '      /=     NE     $/=     {
TRAP            VECR             !      .'     ./=    .NE    .$/=    }
TRAPCHK         WHILE
TRIM            WRITER
TRIMR           XPND
TRUNC           ZEROS
TYPE
```

1.2 Getting Started

In this section we will briefly tell you how to start **GAUSS** , load an ascii data file, and to perform basic operations on the data.

1.2.1 Starting and Exiting GAUSS

GAUSS is started from a DOS prompt by typing

```
gaussi
```

At this point you should see a screen that looks like:

```
D:\GAUSS32>
GAUSS-386i VM  Version 3.2.13  ( Jul 20 1995 )
(C) Copyright 1984-1995 Aptech Systems, Inc. Maple Valley, WA.
All Rights Reserved.
2547688 bytes workspace

>>
```

```
============================================================
Alt-H for help                      L=16   C=3  Path=D:\GAUSS32
```

The prompt '>>' is called the start prompt and is where you may begin typing **GAUSS** statements. The amount of **GAUSS** workspace that is available is given (in bytes). This is the basic memory used by **GAUSS** to hold matrices in memory and to perform **GAUSS** operations. This amount can be controlled by the user in standalone installations. Consult the **GAUSS** documentation for details if you receive an error message denoting a lack of workspace.
Note also that **Alt-H** can be pressed to obtain on-line help. You also given information about which disk drive and directory you are using. To exit **GAUSS** , press **Esc**. You will be asked <y/n> whether you want to exit **GAUSS** . Answering y sends you back to the DOS prompt.

1.2.2 Creating a Data File Using the GAUSS Editor

More information about how to use the **GAUSS** editor is given in a later chapter. At this point we will give you simple instructions about how to create, edit, and save a file containing data that **GAUSS** can use.
At the prompt

```
>>
```

type `edit data1.dat`. You will receive a message indicating that this is a new file that will disappear in a few seconds. After the screen clears you will be in EDIT mode and ready to type data into the file called `data1.dat`. Type in the following data

```
 659  504 11785 12.2
  44   72 22432 12.7
 296  419 13569 12.6
 499  268 10106 12.2
3742 3882 15069 13.2
 460  584 14992 12.8
 648  719 16244 12.6
 123  110 15732 12.5
```

Note, there are spaces between each number, making 8 rows and 4 columns. **GAUSS** will disregard extra spaces between the numbers. To save your work and return to the COMMAND mode, type **F1**.
Now you will create a new file called `first.prg`. From the start prompt type `edit first.prg`. This will open a new file. Type in the following simple program using the editor.

```
/*  This is my first Gauss Program */
load mat[8,4] = data1.dat;  /* loads the ascii data */

    "Data Matrix = " mat;     /* prints the matrix, mat */
    x1 = mat[.,1];            /* puts all rows of column 1 into x1 */
    x2 = mat[.,2:4];          /* puts all rows of columns 2,3,4 in x2 */
    x1tx1 = x1'*x1;           /* inner product of x1 */

"X1 transpose X1 = " x1tx1;
```

Press **F2** to save and execute the program. The output will look like this.

```
Data Matrix =
       659.00000        504.00000        11785.000        12.200000
       44.000000        72.000000        22432.000        12.700000
       296.00000        419.00000        13569.000        12.600000
       499.00000        268.00000        10106.000        12.200000
       3742.0000        3882.0000        15069.000        13.200000
       460.00000        584.00000        14992.000        12.800000
       648.00000        719.00000        16244.000        12.600000
       123.00000        110.00000        15732.000        12.500000
X1 transpose X1 =        15422031.
>>
```

Note that comments can be printed to the screen by enclosing the comment in `" "` as in `"Data Matrix = "`. The number of digits printed can be controlled using the format command (type `format` at the `Help on:` prompt).
If changes need to be made to the program, type `edit first.prg` and you will be returned to the program.

1.2.3 Matrices

You can create matrices from within a program using the **let** statement. Reopen `first.prg` and type the following line in at the end of the program

```
let x = {1 2 3, 4 5 6, 7 8 9};
```

This creates a 3x3 matrix called `x`. To print it out type:

```
print x;
```

As in most programming languages the `=` is the assignment operator. It assigns what's to its right to the symbol on its left. In this case the 3x3 matrix is assigned to the symbol `x`.
The **let** statement can be omitted when a symbol is assigned to a matrix enclosed in curly braces. For example:

```
x = {1 2 3, 4 5 6, 7 8 9};
```

1.2.4 Concatenation

Concatenation is a way to join matrices and vectors together either horizontally or vertically. Of course, in order to do this then the matrices you are trying to join must be conformable. To illustrate, type in and execute the following program.

```
a = { 2.2 1.7 };
b = { -3.2 3.9 };

hc = a~b;
print hc;

vc = a|b;
print vc;
```

The horizontal concatenation operator is `~`. The result of `a~b` is

```
2.2000000       1.7000000       -3.2000000       3.9000000
```

The vertical operator is `|` which stacks conformable matrices and vectors one on top of the other. The result for `a|b` is

```
2.2000000       1.7000000
3.2000000       3.9000000
```

1.2.5 Special Matrices

There are a number of special matrices you can generate in **GAUSS** .

Zeros

To create a nxm matrix of zeros use the command:

```
z = zeros(n,m);
```

Ones

To create a nxm matrix of ones use the command:

```
z = ones(n,m);
```

Normal Random Numbers

To create a nxm matrix of normal random deviates use the command:

```
z = rndn(n,m);
```

Note that $z \sim N(0,1)$.

Identity Matrix

To create an nxn identity matrix use the command:

```
z = eye(n);
```

Additive Sequence

To create an additive sequence of numbers (for instance if you were creating a vector for a time trend) use the following command:

```
z = seqa(1,1,n);
```

The result will be an nx1 vector that begins with 1 and increments by 1 until it reaches n.

1.2.6 Indexing Matrices and Extracting Submatrices

This section describes how to index individual elements of a matrix and extract submatrices. Returning to the example in `first.prg`, recall that we were able to extract a column from the matrix `x` using the command

```
x1 = x[.,1];
```

Similarly, we could have extracted the second row using

```
x3 = x[2,.];
```

The pattern should be apparent. The notation `x[n,m]` refers to the n^{th} row and m^{th} column of `x`. A period `.` in either the row or column position tells **GAUSS** to take all rows or columns, respectively.
Consecutive columns can be extracted using a colon. The following statement extracts all rows of columns 2, 3, and 4 of `x` and puts them into a matrix called `w`.

```
w = x[.,2:4];
```

Sets of rows or columns can be extracted by creating and index using the **let** statement. For instance, to extract rows 2, 5, and 8 you could use

```
idx = { 2 5 8 };
rr = x[idx,.];
```

Note that the **let** statement can be omitted in creating `idx` since it is assigned to numbers enclosed in curly brackets.

Chapter 2

The Editor

In order to run programs in **GAUSS** you'll need a way to send your programs to **GAUSS** for processing. To facilitate this, **GAUSS** includes a full-screen editor that can be used for interactive programming or for submitting batch programs. By default, the **GAUSS** editor is active whenever a program is not being executed. The **COMMAND mode** is used to create short interactive programs. In COMMAND mode you may enter a single line of code at a time and have it execute by pressing ENTER. **EDIT mode** is used to create longer programs that are saved to your disk as files.
Whenever you start **GAUSS** without a command line argument, you are automatically in COMMAND mode. If you Include a command line argument, **GAUSS** assumes that the argument is the name of the file you want to execute. When the program finishes, **GAUSS** returns you to COMMAND mode.

2.1 COMMAND Mode

The COMMAND mode can be used to write short, interactive programs or to submit batch files created in EDIT mode for processing by **GAUSS** . You will know that you are in the COMMAND mode whenever you are greeted by the prompt:

```
>>
```

The prompt '≫' is referred to as the start (program) character. The end program character is '≪'; any statements between these prompts are treated as a program.
Pressing the ENTER key causes a program to begin executing. CTRL-ENTER can be used to drop to the line below without executing the current line. It is possible to toggle off this feature of the editor using the **Alt-F4** key combination. If **Enter** execute is off, then a program is defined as the first block of code, enclosed between the command start and stop characters. For example

```
>>
x=rndn(4,4);
y=sumc(x);
<<
```

The command stop character '≪' is inserted using **F4**. Pressing **F2** begins execution of the program. Before pressing **F2** make sure that your cursor is to the right of or below the stop command '≪'.
Pressing the **F10** key loads the command log file into the editor and places the system in EDIT mode. Previous commands can be executed using **Ctrl-X** or can be cut and pasted to the COMMAND mode screen using block commands. Pressing **F1** restores the screen as it was before the last interactive program was run.
In the tables below are lists of the various key commands available in COMMAND and EDIT modes. Table 2.1 summarizes the commands needed to access on-line help, to select various system options, to access a DOS-shell, and to exit the program and return to DOS. These keyboard commands are available in either the EDIT or COMMAND modes.
In Table 2.2 you will find various commands that enable you to execute programs, edit files, and to toggle various compiler and execute options are summarized. These commands only apply to the COMMAND mode.
The delete, copy, and insert commands found in Table 2.3 can be used in either the COMMAND mode or in the EDIT mode. It is recommended that you learn these in order to save yourself a lot of unnecessary typing.
The search and replace commands are also very useful. These appear in Table 2.4. Although these commands work in the COMMAND mode, you will find them to be most useful when you are editing long programs in the EDIT mode.

2.2 EDIT mode

Files can be created and edited by typing `Edit` followed by the name of the file to be edited. If the file exists, it will be displayed and is ready to edit. If the file does not exist, a message 'New File' will be displayed for a second or two and a blank screen will then appear. The new file can then be edited using the edit commands.
To exit the EDIT mode, press **Alt-X**. A menu will appear and selections can be made by pressing the capitalized letter from one of the following options.

Write	Write the edited file to disk, making a backup having the .BAK extension. Return to COMMAND mode.
Quit	Quit EDIT mode and return to COMMAND mode without saving the file.
e**X**ecute	Saves the file and then executes it.
Debug	Save the file and then execute it using the debugger.
Run Options	Set options for program execution.
Compile Options	Set options for the compiler.

In Table 2.5 various commands are summarized that enable one to execute programs, edit files, and to toggle various compiler and execute options from the EDIT mode.

Alt 1-7	Option menus direct
Alt-C	Option menu tree
Alt-H	Help
Alt-Z	DOS Shell
F7	View error log file
F9	Review last help screen
F10	Edit command log file
Escape	Exit GAUSS
F1	Recall previous screen

Table 2.1: HELP, MENUS, and Miscellaneous Commands

F2	Run command
Enter	Execute command on screen
Alt-F2	Execute last run file, lines off
Ctrl-F2	Execute last run file, lines on
Shift-F2	Execute last edited file
Ctrl-F1	Edit last run file
Ctrl-F3	Edit last output file
Shift-F1	Edit last edited file
Alt-F1	Edit with alternate editor
Ctrl-E	Save screen to file
Ctrl-A	Toggle autoload
Ctrl-L	Toggle GAUSS.LCG on/off
Ctrl-X	Execute block
Ctrl-O	Trace off
Alt-L	Toggle block
Ctrl-V	Toggle compiler trace
Ctrl-W	Toggle DECLARE warnings
Alt-F4	Toggle ENTER execute
Shift-F7	Reset error log file
F3, F4	Command start, stop character

Table 2.2: Command Mode FILE, EXECUTE, and COMPILE Commands

Delete	Delete character, block
Insert	Insert (paste) scrap at cursor
Grey +	Copy line, block to scrap
Grey -	Cut line, block to scrap
Alt-D	Delete line
Alt-K	Delete from cursor to end of line
Ctrl-R	Delete word right
Ctrl-PgDn	Delete all text below cursor
Ctrl-PgUp	Delete all text above cursor
Alt-W	Write block
Alt-P	Print block
Alt-I	Toggle insert mode
Ctrl-N	Insert hard linefeed
Alt-R	Read in file at cursor
Ctrl-Enter	Carriage return, no execute

Table 2.3: DELETE, INSERT, COPY Commands

Alt-G	Go to line
Ctrl-Left	Word left
Ctrl-Right	Word right
Alt-S	Search
Shift-F5	Search next
Alt-T	Translate (replace)
Shift-F6	Translate again
Ctrl-F5	Toggle case sensitivity for search/translate
Ctrl-F6	Toggle escape character recognition for search/translate
Home	Beginning of line-screen
End	End of line-screen
Ctrl-Home	Clear screen
Ctrl-T	Toggle translator

Table 2.4: SEARCH, CURSOR POSITION Commands

Alt-X	Normal editor exit
F1	Save file and exit
F2	Save and execute file, lines on
Alt-F2	Save and execute file, lines off
Alt-O	Rename file
Ctrl-X	Execute block
Alt-F4	Toggle ENTER execute
Ctrl-O	Trace off
Ctrl-V	Toggle compiler trace
Ctrl-W	Toggle DECLARE warnings
Ctrl-A	Toggle autoload
Ctrl-L	Toggle GAUSS.LCG on/off

Table 2.5: Edit Mode FILE, EXECUTE, and COMPILE Commands

Chapter 3

Operators

In this chapter, a brief summary of the **GAUSS** operators is given. Consult the **GAUSS** documentation or the on-line help for additional details.
The basic mathematical, matrix, string array, and string operators are:

```
MATHEMATICAL OPERATORS              MATRIX AND STRING ARRAY OPERATORS
 +   addition                        |       matrix vertical concatenation
 -   subtraction or unary minus     ~       matrix horizontal concatenation
 *   multiplication                  $|      string array vert concatenation
 .*  ExE multiplication              $~      string array horiz concatenation
 ^   ExE exponentiation              '       transpose
 !   factorial                       .'      bookkeeping transpose
 ./  ExE division
 /   division or linear equation solution of Ax = b, for example: x = b/A;
 %   modulo division
 .*. Kronecker product
 *~  horizontal direct product
STRING OPERATORS
$+      string concatenation
```

Symbols used for indexing matrices are: "[", "]", "." and ":". For example,

```
x[1 2 5]        returns the 1st, 2nd and 5th elements of x.
x[2:10]         returns the 2nd through 10th elements of x.
x[.,2 4 6]      returns all rows of the 2nd, 4th, and 6th columns of x.
```

The braces '{' and '}' are used to create matrices. For example, x = { 1 2 3 }.

3.1 Relational Operators

The scalar-returning relational operators are:

```
Less than:                      Not equal:
    z = x < y;                      z = x /= y;
    z = x LT y;                     z = x NE y;
    z = x $< y;                     z = x $/= y;

Greater than:                   Greater than or equal to:
    z = x > y;                      z = x >= y;
    z = x GT y;                     z = x GE y;
    z = x $> y;                     z = x $>= y;

Equal to:                       Less than or equal to:
    z = x == y;                     z = x <= y;
    z = x EQ y;                     z = x LE y;
    z = x $== y;                    z = x $<= y;
```

The result is a scalar 1 or 0, based upon a comparison of all elements of x and y. ALL comparisons must be true for a result of 1 (TRUE).
The "$" is used for comparisons between character data and other nonnumeric data, e.g. NANs.

3.2 Matrix Relational Operators

The matrix-returning relational operators are:

```
Less than:          Not equal:
    z = x .< y;         z = x ./= y;
    z = x .LT y;        z = x .NE y;
    z = x .$< y;        z = x .$/= y;

Greater than:       Greater than or equal to:
    z = x .> y;         z = x .>= y;
    z = x .GT y;        z = x .GE y;
    z = x .$> y;        z = x .$>= y;

Equal to:           Less than or equal to:
    z = x .== y;        z = x .<= y;
    z = x .EQ y;        z = x .LE y;
    z = x .$== y;       z = x .$<= y;
```

The above operators all produce a matrix of 0's and 1's, with a 1 where the corresponding comparison is TRUE.
The "$" is used for comparisons between character data and other nonnumeric data, e.g. NANs.

3.3 Scalar Relational Operators

The logical operators perform logical or Boolean operations on numeric values. On input, a nonzero value is considered TRUE and a zero value is considered FALSE. The logical operators return a 1 if TRUE and a 0 if FALSE.
These operators require scalar arguments. These are the ones to use in If and DO statements:

```
* Complement              * Disjunction           * Equivalence
  z = NOT x;                z = x OR y;             z = x EQV y;

* Conjunction             * Exclusive OR
  z = x AND y;              z = x XOR y;
```

3.4 Matrix Logical Operators

The matrix logical operators perform logical or Boolean operations on numeric values. On input, a nonzero value is considered TRUE and a zero value is considered FALSE. The logical operators return a 1 if TRUE and a 0 if FALSE.

```
* Complement              * Disjunction           * Equivalence
  z = .NOT x;               z = x .OR y;            z = x .EQV y;

* Conjunction             * Exclusive OR
  z = x .AND y;              z = x .XOR y;
```

If the logical operator is preceded by a dot ".", the result will be a matrix of 1's and 0's based upon an element-by-element logical comparison of x and y. For example, if

```
x = { 0 1 0 1 } and y = { 1 1 0 0 }
```

then (x .OR y) will be the vector 1 1 0 1 . Do not use the "." operators in IF or DO...WHILE statements.

Chapter 4

GAUSS Fundamentals

4.1 Precision and Rounding

All calculations in **GAUSS** are done in double precision, with the exception of some of the intrinsic functions, which may use extended precision (18-19 digits of accuracy). Use PRCSN to change the internal accuracy used in these cases. ROUND, TRUNC, CEIL and FLOOR convert floating point numbers into integers. The internal representation for the converted integer is still 64-bit (double precision).
Each matrix element in memory requires 8 bytes of workspace. See the function CORELEFT to determine availability of workspace.

```
-------------------------------------------------------------------------
      BASE10            Convert number to #.### and a power of 10.
      CEIL              Round up towards +INF.
      FLOOR             Round down towards -INF.
      PRCSN             Set computational precision for matrix operations.
      ROUND             Round to the nearest integer.
      TRUNC             Truncate decimal portion of number.
-------------------------------------------------------------------------
```

4.2 Conditional Branching

Conditional branching is done with the IF statement:

```
if x > 0;
    rv = 1;
    print "Positive";
elseif x < 0;
    rv = -1;
    print "Negative";
else;
```

```
            rv = 0;
            print "Zero";
        endif;
```

The expression after the IF or the ELSEIF must be a scalar-returning expression.
Use the relational and logical operators without the dot.
ELSEIF and ELSE are optional. There can be multiple ELSEIF's.

4.3 Unconditional Branching

Unconditional branching is done with GOTO.

```
Examples:
/* coin toss... */
toss:
    coin = rndu(1,1);
    if coin > .49 and coin < .51;
        goto edge;
    elseif coin >= .51;
        heads = heads + 1;
    endif;
    t = t + 1;
    goto toss;
edge:
    print "It's on edge!";
    print "H " heads " T " t-heads;
```

```
/* file check... */
open f1 = mydat for read;
if f1 == -1;
   goto errout( "File not found", -1 );
endif;
   .
   .
errout:
   pop rv;
   pop msg;
   errorlog msg;
   _errval = rv;
   end;
```

The target of a GOTO is called a label. Labels must begin with '_' or an alphabetic character and are always followed by a colon.
GOTO, like GOSUB, can pass arguments via the stack. If arguments are passed, they are retrieved (POPed) in the reverse order they are passed.

4.4 Looping

Looping is controlled with the DO statement.

```
do while st > tol;      /* loop if true */
   .
   .
   .
endo;


do until st <= tol;      /* loop if false */
   .
   .
```

```
    .
endo;
```

```
BREAK;                  Jump to the statement following ENDO.
CONTINUE;               Jump to the top of a DO loop.
```

4.5 Subroutines

Subroutines are marked by labels. GOSUB branches to a subroutine label and begins execution. RETURN returns from a subroutine to the line following the GOSUB. Arguments can be passed to and from the subroutine via the stack. They are retrieved (POPed off the stack) in the reverse order they are passed.

```
gosub fopen( "MYFILE", "r" );
pop fp;
  .
  .
fopen:
    pop _fmode;
    pop _fname;
    if _fmode $== "r";
       open _fp = ^_fname for read;
    elseif _fmode $== "w";
       create _fp = ^fname with x,1,8;
    elseif _fp $== "a";
       open _fp = ^_fname for append;
    else;
       _fp = -1;
    endif;
    return( _fp );
```

4.6 Procedures

A procedures in **GAUSS** is a user-defined function that can be used as if it were an intrinsic part of the language. They can be small and simple or large and complicated and can be used to modularize your **GAUSS** programs. You can build your own procedures to compute Least Squares, do Seemingly Unrelated Regression, or any other econometric procedure that you use repeatedly. In order to write and use procedures effectively, you will need to consult the Procedures and Keywords chapter in the **GAUSS** documentation. To give you some idea of what is involved, we present the basic elements and give an example below.
The key elements of a procedure are:

- A procedure declaration

- Local variable declarations
- Body of the procedure
- Return from procedure
- End of procedure

There is always one **PROC** statement and one **LOCAL** statement in a procedure. Any statements that come between these two statements are part of the procedure. An example of a procedure is:

```
PROC (3) = crosprod(x,y);
     LOCAL r1, r2, r3;
     r1 = x[2,.].*y[3,.]-x[3,.].*y[2,.];
     r2 = x[3,.].*y[1,.]-x[1,.].*y[3,.];
     r3 = x[1,.].*y[2,.]-x[2,.].*y[1,.];
     RETP( r1,r2,r3 );
ENDP;
```

The "(3) = " indicates that the procedure returns three arguments. All local variables, except those listed in the argument list, must appear in the **LOCAL** statement. The local variables cannot be carried out of the procedure. The **RETP** statement captures the computations that can be carried out of the procedure. The **ENDP** is necessary and must appear at the end of the procedure definition. Procedure definitions cannot be nested.
To use the PROC, it must first be compiled. This is accomplished by pressing **F2** while in EDIT mode (assuming you have the file containing the procedure open) or by issuing the `run` command from the COMMAND mode. That is

```
run my.prc;
```

where `my.prc` is the name of the file in which you saved the procedure.
The procedure may now be used by your **GAUSS** programs. The syntax for using this example function is:

```
{ a1,a2,a3 } = crosprod(u,v);
```

The results are assigned to the variables called `a1`, `a2`, and `a3` which are the crossproducts computed in the procedure.

Chapter 5

Linear Statistical Models

5.1 Introducton

In this Chapter the fundamentals of the Classical Linear Regression Model (CLRM) are considered. The chapter begins with a simple linear model for the mean of a population, evolves into a 2-variable regression model and then to the Multiple regression model. This handbook will present the **GAUSS** commands that replicate numerical examples, and thus providing a basis for all linear model estimation.

5.2 Linear Statistical Model 1

In this section it is noted that the model given in (3.2.2a), in which a random variable has a mean and varinace, is a linear model. Familiar assumptions are put into vector and matrix notation. It is important to master this notation, as it will be the basis for all future work.

5.3 Linear Statistical Model 2

In this section computations associated with the Example in Chapter 5.3.6 of *ITPE2* will be carried out. The data on Consumption and Income in Table 5.1 in the text is contained in the file TABLE5.1 on the disk available with this book. The regression model relates consumption to income.
First, LOAD the data, which is in an ASCII file. Insert a column of ones in the first column to represent the "intercept" variable. Print and examine the data to confirm it is identical to that in the text.

```
load dat[20,2] = table5.1;
y = dat[.,1];
x = dat[.,2];
x = ones(20,1)~x;
```

```
format 8,4;
y~x;
```

You now have the data set represented by the vector y and the matrix X but you do not know the true parameters, beta, or the variance of the error term, σ^2. We need to solve the system of linear equations (5.3.8b) represented by $x'x * b = x'y$. Use Equation (5.3.11). There are several **GAUSS** functions that invert matrices. See your **GAUSS** manual for a description of INV, INVPD and SOLPD.
To show the intermediate results follow the sequence of steps outlined in Equations (5.3.22c and 5.2.23. First form the inverse of $X'X$ and $X'y$ and print.

```
ixx = inv(x'x);
xty = x'y;
format 14,10;
ixx~xty;
```

Note that $X'y$ is named XTY instead of XY to avoid confusion with the **GAUSS** QUICK GRAPHICS proc. Now compute the least squares estimates.

```
b = ixx*xty;
format 8,5;
b';                                    /*  Eq. 5.2.23  */
```

The most efficient way to solve the equations in **GAUSS** is to use the matrix / operator. That is, $b = inv(x'x) * (x'y) = (x'y)/(x'x) = y/x$.

```
 b = y/x;
 b';
```

Compute the least squares residuals and the sum of squared errors.

```
ehat = y - x*b;
sse = ehat'ehat;
sse;
```

Next compute the degrees of freedom. In general, T is the number of rows in the matrix x and k is the number of columns in x. The degrees of freedom is the difference.

```
t = rows(x);
k = cols(x);
df = t - k;
df;
```

The estimate of the error variance σ^2 is sse/df.

```
sighat2 = sse/df;
sighat2;                               /*  Eq. 5.3.25  */
```

Estimate the variance-covariance matrix of the LS estimator. The function invpd takes the inverse of a positive definite matrix, and is more efficient than inv for these types of matrices.

```
ixx = invpd(x'x);
covb = sighat2*ixx;
covb;                                   /*  Eq. 5.3.26  */
```

Let x0 represent a matrix of subsequent values of explanatory variables that we would like to use for prediction purposes. It contains 1 observation.

```
let x0[1,2] = 1 22;
```

The predicted values for y using those values of x are:

```
y0 = x0*b;
y0;                                     /*  Eq. 5.3.28  */
```

The estimated covariance matrix for the prediction error is

```
y0var = sighat2*(x0*ixx*x0' + 1);
y0var;                                  /*  Eq.  5.3.17b  */
```

The square roots of the diagonal elements are the standard errors for the prediction errors.

```
std = sqrt(diag(y0var)); std;
```

The coefficient of determination R^2 can be computed using the sum of squared errors (SSE computed above) and the total sum of squares (SST).

```
ybar = meanc(y);
sst = y'y - t*ybar^2;
r2 = 1 - (sse/sst);
r2;
```

The adjusted R^2, $\bar{R}^2$, is:

```
rbar2 = 1 - (t-1)*(1-r2)/(t-k);
rbar2;
```

5.4 The General Linear Statistical Model Model 3

In this Section of *ITPE2* the notation for the CLRM with K explanatory variables is presented. Carefully analyze all the variations on this basic notation.

5.4.1 Point Estimation

In this Section we repeat many of the steps taken in Chapter 5.3. The data on a poultry production function is given in Table 5.3 in *ITPE2*. The dependent variable, y, is the average weight of the chickens at a point in time and the explanatory or independent variables are cumulative food consumption and its square.
Create the y vector and X matrix and print the data.

```
load dat[15,2] = table5.3;
y = dat[.,1];
xvec = dat[.,2];
x = ones(15,1)~xvec~(xvec^2);
y~x;
```

Compute the cross-product matrices between x and itself and x and y and print. These matrices appear in Equation (5.5.22b)

```
xx = x'x;
xty = x'y;
xx~xty;
```

Compute and print the matrix inverse of x'x. Compare to (5.5.23)

```
ixx = invpd(x'x);
ixx;
```

Compute the sample estimates using (5.5.24)

```
b = ixx*xty;
b';
```

Note again that computationally it is more efficient to use " / "

```
b = y/x;
b';
```

Now plot the estimated production function for the sample values of X.

```
yhat = x*b;
library qgraph;
xy(xvec,yhat);
```

5.5 Sampling Properties of the Least Squares Rule

In this Section the mean and covariance matrix of the least squares estimator are determined. In Section 5.10 a Monte Carlo Experiment is carried out that explores these properties.

5.5.1 Sampling Properties–The Gauss-Markov Result

In this Section it is proved that the least squares estimator is the Best Linear Unbiased Estimator (BLUE) under the assumptions of the CLRM.

5.5.2 Estimating the Scale Parameter σ^2

In order to estimate the error variance σ^2 calculate the sum of squared least squares residuals. Note that an alternative computational form is given in Equation (5.8.7)

```
ehat = y - x*b;
sse = ehat'ehat;
sse;
```

Calculate the degrees of freedom.

```
t = rows(x);
k = cols(x);
df = t - k;
df;
```

Calculate the parameter estimate using (5.8.7)

```
sighat2 = sse/df;
sighat2;
```

Calculate the estimate of the covariance matrix of b.

```
cov = sighat2*ixx;
cov;
```

5.5.3 Prediction and Degree of Explanation

Specify the following X0 matrix, as in Section 5.9.2 of *ITPE2*.

```
let x0[1,3] = 1 10  100;
t0 = rows(x0);
```

The corresponding predicted value of y is

```
y0hat = x0*b;
y0hat';
```

The sampling variance of the forecast error for y0hat as an estimator of y0 is given in (5.9.3a)

```
y0var = sighat2*(x0*ixx*x0' + eye(t0));
y0var;
```

The sampling variance of y0hat as an estimator of E(y0) is given in (5.9.4a)

```
Ey0var=sighat2*(x0*ixx*x0');
Ey0var;
```

The degree of explanation by the linear model is given by R^2 in (5.9.9). The total sum of squares SST is

```
ybar = meanc(y);
sst = y'y - t*(ybar^2);
sst;
```

The value of R^2 is

```
r2 = 1 - (sse/sst);
r2;
```

and the Adjusted-R^2 is

```
rbar2 = 1 - (t-1)*(1-r2)/(t-k);
rbar2;
```

5.5.4 OLS Proc

For future reference, and use, combine all that you have learned into an Ordinary Least Squares (OLS) procedure. Save this procedure in a separate file MYOLS.G in the \SRC subdirectory so that you may call it later and it will be LOADed automatically. **GAUSS**'s PROC OLS can also be used. See your **GAUSS** manual for details

```
proc(2) = myols(x,y);
   local *;

   b = y/x;
   t = rows(x);
   k=cols(x);
   df = t - k;
   ehat = y - x*b;
   sse = ehat'ehat;
   sighat2 = sse/df;
   covb = sighat2*invpd(x'x);
   ybar = meanc(y);
   sst = y'y - T*(ybar^2);
   r2 = 1 - (sse/sst);
   rbar2 = 1 - (t-1)*(1-r2)/df;

   format 8,2;
   "Number of observations: " T;
```

```
    "Degrees of freedom:     " df;

    format 10,5;
    "Sum of Squared Errors: " sse;
    "Total Sum of Squares:  " sst;
    "R-squared:             " r2;
    "R-bar-squared:         " rbar2;
    "Sigmahat^2:            " sighat2;
    "Standard Error:        " sqrt(sighat2);
     ?;
    "Estimated coefficients: " b';
     ?;
    "Variance-Covariance Matrix for b: ";
     covb;


  retp(b,covb);
endp;
```

The beauty of this OLS program is that you understand every step of its workings, unlike some "canned" programs you may use.
After you have created the file containing the PROC run it using the F2 key which will compile the program and check for errors. Then you may use it as

```
{b,covb} = myols(x,y);
```

5.6 A Monte Carlo Experiment to Demonstrate the Sampling Performance of the Least Squares Estimator

To better understand the properties of the estimators, run a series of Monte Carlo experiments. To do this, write a procedure to simulate the data generation process of a linear statistical model. Create many data sets that follow this process, the only difference being the values of the random disturbances. Then estimate the parameters of the model for each data set and observe how the parameter estimates are distributed. They should agree closely with the theoretical properties that have been derived.
The sampling experiment will use the parameter values in Equation (5.10.1) and the design matrix X in (5.10.2), which is in the data file JUDGE.X on the data disk. In these simulations, assume that the random errors have a uniform distribution with mean zero and variance 2. While the **GAUSS** random number generators could be used to create the values of the random disturbances, use instead the "official" uniform random numbers contained in the data file URANDOM.DAT. This data set is a **GAUSS** data file and may be read using the READR command. It contains 10,000 uniform random numbers falling in

the (0,1) interval. Since these random numbers have mean 1/2 and variance 1/12 we must transform them to mean 0 and variance 2 by subtracting 1/2 and multiplying by sqrt(24).
Start "small" and create NSAM = 10 samples of size T = 20 and obtain the least squares estimates of β and σ^2. First create a (T x NSAM) matrix E containing the random errors.

```
t = 20;                          /*  sample size          */
k = 3;                           /*  number of regressors */
nsam = 10;                       /*  number of samples    */
nr = t*nsam;                     /*  number obs. to read  */
open f1 = urandom.dat;           /*  open data set        */
e = readr(f1,nr);                /*  read nr obs.         */
f1 = close(f1);                  /*  close file           */
```

The uniform random disturbances are now in a (NR x 1) vector. To create the desired (T x NSAM) matrix E use the RESHAPE function. RESHAPE stores the data in "row major" form so form an (NSAM x T) matrix and then transpose it.

```
e = (reshape(e,nsam,t))';
```

Now correct the mean and variance.

```
e = sqrt(24)*(e - 0.5);
```

Now LOAD `x` and define `beta`.

```
load x[t,k]=judge.x;
let beta = 10.0 0.4 0.6;
```

The matrix of error terms created, `e`, has T rows — the size of each separate sample, and NSAM columns — the number of samples. Because of the special features of matrix addition in **GAUSS**, when this matrix is added to the vector `x*beta`, a matrix of y's is created. The first column is equal to `x*beta` plus the first column of `e`, the second column is equal to `x*beta` plus the second column of `e`, etc.
The same formula for computing the estimated coefficients, `y/x` can still be used. The first column will contain the estimated coefficients for the first sample, the second column the second sample, etc. The `sighat2` is a column vector containing the estimated variances for each sample. Since we will use it several times, create PROC MC to carry out the calculations. The arguments are `x`, `beta` and a (T x NSAM) matrix of random disturbances `e`.

```
proc (2) = mc(x,beta,e);
local *;

     y = x*beta + e;
```

```
        t = rows(x);
        k = cols(x);
        b = y/x;

        ehat = y-x*b;
        sighat2 = sumc(ehat^2)/(t-k);
        retp(b,sighat2);

endp;
```

The arguments of PROC MC are the design matrix `x`, the beta vector, and the matrix of random disturbances `e`. The procedure will return the (K x NSAM) matrix of estimated coefficients and (NSAM x 1) vector of estimated variances. Run the procedure to compile it and then use it to carry out the Monte Carlo exercise.

```
{b,sighat2} = mc(x,beta,e);
b'~sighat2;
```

Compare the values of the estimated parameters to those in Table 5.4. They should be identical. Note that the parameter estimates vary from sample to sample.

Now carry out the Monte Carlo experiment on a grander scale. Use NSAM = 500 samples of size T = 20. Unfortunately, due to limits on the abilities of personal computers, a matrix in **GAUSS** can only hold 8190 elements. Thus the 10,000 random numbers cannot all be read into a single matrix. Thus we will divide the computations into two steps, using NSAM = 250 each time. First, construct the matrices of uniform random errors with zero mean and variance one. SAVE them for future use. They will be given the extension .FMT and can be LOADed when needed.

```
t = 20;
k = 3;
nsam = 250;
nr = t*nsam;
open f1 = urandom.dat;
e1 = readr(f1,nr);                  /*  read 1st 5000 obs.   */
e2 = readr(f1,nr);                  /*  read 2nd 5000 obs.   */
f1 = close(f1);

e1 = (reshape(e1,nsam,t))';         /*  250 random vectors   */
e2 = (reshape(e2,nsam,t))';         /*  250 random vectors   */

e1 = sqrt(12)*(e1-0.5);             /*  adjust to U(0,1)     */
e2 = sqrt(12)*(e2-0.5);

save e1uni = e1;                    /*  SAVE to E1.FMT       */
save e2uni = e2;                    /*  SAVE to E2.FMT       */
```

Now, assuming `x`, `beta`, and PROC MC are still in memory, transform the errors to have variance 2 and execute the Monte Carlo.

```
e1 = sqrt(2)*e1;
e2 = sqrt(2)*e2;

{b1,s1} = mc(x,beta,e1);
{b2,s2} = mc(x,beta,e2);
```

Stack all the parameter estimates into a matrix and clear unneeded matrices.

```
est=(b1|s1')~(b2|s2');
e1=0; e2=0; b1=0; b2=0; s1=0; s2=0;
```

Next write another procedure to summarize the results of the Monte Carlo experiment. Call it MCSUM. It takes as its arguments `param`, which is the matrix of parameter estimates, the matrix of explanatory variables, `x`, the vector of true coefficients, `beta`, and the true variance of the error terms, `sigma2`.

```
proc mcsum(param,x,beta,sigma2);
     local *;

        format 8,5;
        "True parameters:    "  beta';; sigma2;
        "Mean of estimates:  "  meanc(param')';
        "Max of estimates:   "  maxc(param')';
        "Min of estimates:   "  minc(param')';
        "True variances:     "  diag( sigma2*inv(x'x) )';
        "Estimated variances:" (stdc(param')^2)';

        format 8,2;
        "Sample Size:        "  rows(x);
        "Number of Samples:  "  cols(param);
         retp("");
     endp;
```

Print out a summary of the results of your Monte Carlo experiment.

```
mcsum(est,x,beta,2);
```

If you don't get a nice table of results check to make sure everything is typed as it should be, including the transpose operators. Your results should be identical to those reported in Chapter 5.10.2 in *ITPE2*.
Use PROC HISTP to obtain histograms for each set of parameter estimates.

```
{ c1,m1,freq1 } = histp(est[1,.]',30);
{ c2,m2,freq2 } = histp(est[2,.]',30);
{ c3,m3,freq3 } = histp(est[3,.]',30);
{ c4,m4,freq4 } = histp(est[4,.]',30);
```

Increase the sample size to T=40, and the number of samples to NSAM = 250 and repeat the exercise. What happens to the average parameter estimates and their variances?

Chapter 6

The Normal General Linear Model

In this Chapter of *ITPE2* the linear statistical model of Chapter 5 is analyzed with the additional assumption that the random disturbances follow a normal distribution.

6.1 Maximum Likelihood Estimation

The maximum likelihood estimators of the parameters β and σ^2 of the normal linear model are presented in Sections 6.1-6.1.4. We begin by examining the Monte Carlo results in Section 6.1.5. These simulations will use the 10,000 random numbers in the **GAUSS** data file NRANDOM.DAT. In the experiments the variance of the disturbance term is to be .0625. Thus the N(0,1) random numbers will be multiplied by `sqrt(.0625)`. The X matrix is the same one used in Chapter 5 and is in the file JUDGE.X. The true betas are 10.0, 0.4 and 0.6 as in Chapter 5.
First, create one sample of size T = 20, print out the sample values y, the N(0,1) deviates and X, as in Equation 6.1.27.

```
t = 20;                          /*  number of obs.       */
k = 3;                           /*  number of regressors */
nsam = 1;                        /*  number of samples    */
nr = t*nsam;                     /*  total obs. to read   */
open f1 = nrandom.dat;           /*  open file            */
e = readr(f1,nr);                /*  read nr obs.         */
f1 = close(f1);                  /*  close file           */
u1 = (reshape(e,nsam,t))';       /*  u1 is (T x NSAM)     */
e1 = sqrt(.0625)*u1;             /*  adjust variance      */
load x[t,k] = judge.x;           /*  load X               */
let beta  =  10.0 0.4 0.6;       /*  true beta            */
y1 = x*beta + e1;                /*  create y             */
```

```
format  9,6;                        /*  print               */
y1~u1~x;                            /*  Eq. 6.1.27          */
```

Now carry out a Monte Carlo experiment using NSAM = 10 samples. Do the Monte Carlo experiments using PROC MC written for Chapter 5. You must place the proc in memory or it must be in a file (MC.G) that can be found by the **GAUSS** auto-loading feature.

```
nsam = 10;                          /*  number of samples   */
nr = t*nsam;                        /*  number of obs.      */
open f1 = nrandom.dat;              /*  open file           */
e = readr(f1,nr);                   /*  read nr obs.        */
f1 = close(f1);                     /*  close file          */
u = (reshape(e,nsam,t))';           /*  u is (T x 10)       */
e = sqrt(.0625)*u;                  /*  adjust variance     */

{b,sighat2} = mc(x,beta,e);         /*  execute monte carlo */
```

Print out the estimates and compare them to Table 6.1 in *ITPE2*.

```
format 8,5;
b'~sighat2;
```

Calculate the true covariance matrix and examine it.

```
ixx = invpd(x'x);
ixx;                                /*  Eq. 6.1.28          */

truecov = .0625 * ixx;
truecov;                            /*  Eq. 6.1.30          */
```

Calculate the estimated variances of the parameter estimators for these samples and compare the true and estimated variances to those in Table 6.2.

```
truevar = diag(truecov);
estvar  =  diag(ixx) .* sighat2';
truevar';
?;
estvar';                            /*  Table 6.2          */
```

Now carry out the experiment using NSAM = 500 samples, breaking the computations into two parts as we did in Chapter 5. First construct, and SAVE, the matrices of N(0,1) random numbers in this convenient form for later use. They will be given a .FMT extension and can be LOADed when needed in later chapters.

```
nsam = 250;                         /*  number of samples   */
nr = t*nsam;                        /*  obs. to read        */

open f1 = nrandom.dat;              /*  open file           */
e1 = readr(f1,nr);                  /*  read 250 samples    */
```

```
e2 = readr(f1,nr);                  /*  read 250 samples     */
f1 = close(f1);                     /*  close file           */

e1 = (reshape(e1,nsam,t))';         /*  e1 is (T x 250)      */
e2 = (reshape(e2,nsam,t))';         /*  e2 is (T x 250)      */

save e1nor  =  e1;                  /*  saved to e1nor.fmt   */
save e2nor  =  e2;                  /*  saved to e2nor.fmt   */
```

Transform the errors to have the desired variances

```
e1 = sqrt(.0625)*e1;                /*  change variances     */
e2 = sqrt(.0625)*e2;
```

Execute the Monte Carlo.

```
{b1,s1}  =  mc(x,beta,e1);          /*  execute monte carlo  */
{b2,s2}  =  mc(x,beta,e2);
```

Stack the regression parameter estimates into one (K x NSAM) matrix and the estimates of the error variance into a (1 x NSAM) vector for use throughout the rest of this chapter.

```
b = b1~b2;                          /*  b is (K x 500)       */
sighat2 = s1'~s2';                  /*  sighat2 is (1 x 500) */
```

Stack all the estimates into one matrix and clear memory of unneeded matrices.

```
param = b|sighat2;

e1 = 0; e2 = 0;                     /*  clear memory         */
b1 = 0; b2 = 0;
s1 = 0; s2 = 0;
```

Print out a summary of the Monte Carlo results using PROC MCSUM written in Chapter 5, and compare the results to those on page 232 of *ITPE2*.

```
mcsum(param,x,beta,.0625);          /*  summarize results    */
```

Use **GAUSS**'s QUICK GRAPHICS to construct histograms similar to Figures 6.1 - 6.4 in *ITPE2*. Use 30 intervals. Your graphs will not look exactly like those in text as **GAUSS** will form partitions of the data that are not the same as those in the text.

```
library qgraph;
{ c1,m1,freq1 } = histp(b[1,.]',30); /*  Figure 6.1          */

{ c2,m2,freq2 } = histp(b[2,.]',30); /*  Figure 6.2          */

{ c3,m3,freq3 } = histp(b[3,.]',30); /*  Figure 6.3          */
```

The histogram for the estimated variances uses the Chi-squared random variable $(T-K)*\hat{\sigma}^2/.0625$.

```
chi2var = (t - k)*(sighat2')/.0625;
{ c4,m4,freq4 } = histp(chi2var,30); /*  Figure 6.4        */
```

6.2 Restricted Maximum Likelihood Estimation

In this Section exact linear restrictions are imposed on the Maximum Likelihood or Least Squares estimators and the consequences studied. Write a procedure, PROC RLS, that computes the restricted least squares parameter estimates given `b`, the least squares estimates, `r` the (J x K) information design matrix, `rr` a (J x 1) vector of known constants (the right-hand-side of Equation 6.2.1, and the design matrix `x`.

```
proc  rls(b,r,rr,x);
    local *;
    ixx = invpd(x'x);
    q = invpd(r*ixx*r');
    bstar = b + ixx*r'*q*(rr-r*b);
    retp(bstar);
endp;
```

Use PROC RLS to calculate restricted least squares estimates for the example in Section 6.2.3 in *ITPE2* using the NSAM = 500 ML/LS estimates `b` from the Monte Carlo experiment in Section 6.1.

```
let r[1,3] = 0 1 1;
rr = 1;
bstar = rls(b,r,rr,x);
```

Calculate the mean values of the RLS estimates and compare them to the true values. The average value of the restricted intercept in the text is incorrect due to a typographical error.

```
format 8,4;
meanc(bstar');
```

Calculate the average value of the estimated RLS covariance matrices. As an estimator of the error variance used the unbiased estimates `sighat2`. Use a DO-LOOP to carry out the computations. To speed up execution compute the "fixed" part of the covariance matrix (See Equation 6.2.17) outside the loop.

```
c  =  ixx - ixx*r'*invpd(r*ixx*r')*r*ixx;
ind = 1;
covbstar = zeros(k,k);

do while ind le 500;
covbstar = covbstar + (sighat2[1,ind] .* c)/500;
ind = ind+1;
endo;
```

Print out the average covariance matrix and compare it to (6.2.22).

```
covbstar;
```

Compute the true covariance matrix of the restricted least squares estimator and compare it to the average.

```
covrls =  .0625 * c;
```

Finally compute the "empirical" estimator variances and compare them to the true and average values.

```
stdc(bstar')^2;
```

6.3 Interval Estimation

6.3.1 Single Linear Combination of the Beta Vector

For interval estimation the inverse of the cumulative distribution function is typically needed. Write a function to estimate the inverse of the Student's-t distribution. Within the function a sequence of t-values will be computed, and the value closest to the specified alpha will be found using the MININDC function. The corresponding t-value is then computed.

```
fn invt(df,alpha)  = .995 +
 .005*minindc( abs( cdftc(seqa(1,.005,500),df) - alpha));
```

Use the function to compute the t-statistic for α level = 0.025 and T - K degrees of freedom with T = 20 and K = 3, and check it using CDFTC.

```
tstat  =  invt(t - k,.025);
tstat cdftc(tstat,t - k);
```

Compute the 95(6.1.26) using the estimates b from the 500 Monte Carlo samples. See Equation 6.3.7b. Note that these statements compute the intervals for all 500 samples but can be used for a single sample as well.

```
akk  =  sqrt(diag(ixx));
width  =  (tstat * akk) .* sqrt(sighat2);
lb  =  b - width;
ub  =  b + width;
```

Print out the point estimates and confidence intervals for β_1 and β_2 for the first 10 samples, as in Table 6.3

```
((b[1,1:10]|lb[1,1:10]|ub[1,1:10])');  /*  table 6.3 */

((b[2,1:10]|lb[2,1:10]|ub[2,1:10])');
```

Compute the percent of the 500 samples in which the true value of the parameter is contained within the interval for each parameter.

```
within  =  (lb .<=  beta) .and (ub .>=  beta);
meanc(within')';
```

6.3.2 Two or More Linear Combinations of the Beta Vector

We will compute and graph a joint confidence interval for β_2 and β_3 using PROC CONFID written in Chapter 3.
To compute the necessary F-statistic write a function to compute the inverse of the F-distribution c.d.f.

```
fn invf(df1,df2,alpha) = 0.95 + 0.05*
                    minindc( abs (
                    cdffc( seqa(1,.05,2000),df1,df2)
                                          - alpha ));
```

Use the function to compute the .05 critical value for 2 and T - K degrees of freedom, and check it using CDFFC.

```
fstat  =  invf(2,t - k,.05);
fstat cdffc(fstat,2,t - k);
```

Use the estimates from the first Monte Carlo sample and compute the values needed for the joint confidence interval for β_2 and β_3.

```
b1 = b[.,1];
sig1 = sighat2[1,1];
cov1 = sig1*ixx;

let pos  =  3 2;
d  =  confid(b1,cov1,fstat,pos);
```

So that the scale of the graph will be consistent with that in the text use the QUICK GRAPHICS function SCALE.

```
library qgraph;
let xx = 0 1.1;
let yy = 0 1.1;
scale(xx,yy);
```

Graph the confidence ellipse.

```
xy(d[.,1],d[.,2]);
```

Reset the default QUICK GRAPHICS parameter values using GRAPHSET.

```
graphset;
```

6.3.3 Interval Estimation of σ^2

Write a function to compute the inverse of the Chi-square distribution.

```
fn invchi(df,alpha) = 0.95 + 0.05*
                      minindc( abs(
                      cdfchic(seqa(1,.05,2000),df)
                                          - alpha));
```

Compute the critical values of the Chi-square distribution with T - K degrees of freedom for alpha = .025 and .975 and check them.

```
chi1  =  invchi(t - k,.025);
chi2  =  invchi(t - k,.975);
chi1 chi2;
cdfchic(chi1,t - k) cdfchic(chi2,t - k);
```

Compute the confidence intervals for σ^2 (See Equation 6.3.15) and print out the point estimates and intervals for the first 10 samples, as in Table 6.4.

```
lb  =  (t - k)*sighat2/chi1;
ub  =  (t - k)*sighat2/chi2;

(sighat2[1,1:10]|lb[1,1:10]|ub[1,1:10])';   /*  table 6.4  */
```

Compute the percent of the 500 intervals that contain the true value of σ^2.

```
within  =  (lb .<=  .0625) .and (ub .>=  .0625);
meanc(within');
```

6.3.4 Prediction Interval Estimator

Compute the 95% prediction interval for y_0 if $x_1 = x_2 = x_3 = 1$. See Equation 6.3.21 in the text. Use the first Monte Carlo sample.

```
let x0  =  1 1 1;
tstat  =  invt(t - k,.025);
width  =  tstat*sqrt(x0'ixx*x0 + 1) .*  sqrt(sighat2[1,1]);
y0hat  =  x0'b[.,1];
```

Print out the lower bound, point estimate and upper bound.

```
(y0hat-width)~y0hat~(y0hat+width);
```

6.4 Hypothesis Testing

6.4.1 The Likelihood Ratio Test Statistic

Test the joint hypotheses that $\beta_2 = 0.4$ and $\beta_3 = 0.6$. First construct the matrix `r` and the vector `rr` that describe the constraint to test.

```
let r[2,3]  =  0  1  0
               0  0  1;

let rr  =  0.4 0.6;
```

Write the function LAMBDA to compute the F-statistic for a given `r`, `rr`, `b` and `ixx` (the inverse of $X'X$). See Equation 6.4 7 in the text.

```
fn lambda(r,rr,b,ixx,sighat2)  =
    (r*b - rr)'inv(r*ixx*r')*(r*b - rr)/(rows(rr)*sighat2);
```

Compute the F-statistic and its significance level for the first Monte Carlo sample.

```
lam  =  lambda(r,rr,b[.,1],ixx,sighat2[1,1]);
j  =  rows(rr);
lam cdffc(lam,j,t - k);
```

What conclusion about the hypothesis would you make on the basis of this sample of data?
The function LAMBDA computes the value of the likelihood ratio test statistic for a single sample of data. In order to calculate the values of the test statistic for all 500 Monte Carlo samples use a DO-LOOP.

```
lam = zeros(500,1);                    /*  storage matrix       */

ind = 1;                               /*  begin loop           */
do while ind le 500;
lam[ind,1]  =  lambda(r,rr,b[.,ind],ixx,sighat2[1,ind]);
ind  =  ind + 1;
endo;
```

Print out the values for the first 11 samples and compare with the 5value of a F(2,17) distribution, 3.59.

```
lam[1:11,1]';
```

Compute the percent of test statistic values that are less than or equal to 3.59

```
meanc(lam .<=  3.59);
```

Plot the histogram for the test statistics as in Figure 6.9.

```
v  =  seqa(1,1,7);
{ c,m,freq } = histp(lam,v);
```

Compute the percent of test statistic values that fall in the intervals [0,1], (1,2], ..., (6,7]. The percentages reported in the text appear to be incorrect.

```
(freq'/500);
```

6.4.2 A Single Hypothesis

A t-test is used to test a single hypothesis. Write a function to compute the test statistic given in Equation 6.4.25.

```
fn ttest(r,rr,b,ixx,sighat2)  =

          (r*b - rr)./sqrt(sighat2*r*ixx*r');
```

Test the hypothesis that β_1 is 10 using the first Monte Carlo sample.

```
let r[1,3]  =  1 0 0 ;
rr  =  10;
ts  =  ttest(r,rr,b[.,1],ixx,sighat2[1,1]);
ts cdftc(ts,t - k);
```

Test the hypothesis that the sum of β_2 and β_3 is one.

```
let r[1,3]  =  0  1 1 ;
rr  =  1;
ts  =  ttest(r,rr,b[.,1],ixx,sighat2[1,1]);
ts cdftc(ts,t - k);
```

Do you reject, or not, the hypothesis?
Perform t-tests for the hypotheses that each of the parameters are equal to their true values for the 500 Monte Carlo samples.

```
stderr  =  sqrt(diag(ixx) .* sighat2);
tval  =  (b - beta) ./ stderr;
```

Print out the t-values for the first 10 samples.

```
tval[.,1:10]';
```

Compute the percent of test values that exceed the 5% critical value, 2.11.

```
meanc((abs(tval) .>=  2.11)')';
```

Graph a histogram of the test statistic values for each parameter and compare to Figures 6.10 - 6.12. Once again your graphs will not be exactly the same as those in the text due to different partitioning of the horizontal axis.

```
{ c1,m1,freq1 } = histp(tval[1,.]',30);
{ c2,m2,freq2 } = histp(tval[2,.]',30);
{ c3,m3,freq3 } = histp(tval[3,.]',30);
```

6.4.3 Testing a Hypothesis about σ^2

Test the hypothesis that $\sigma^2 = .0625$, using Equation 6.4.28 in the text for the first Monte Carlo Sample.

```
chistat  =  (t - k)*sighat2[1,1]/.0625;
chistat cdfchic(chistat,t - k);
```

Repeat the test for all the Monte Carlo samples and print out the test statistic values for the first 10 samples.

```
chistat  =  (t - k)*sighat2'/.0625;
chistat[1:10,1];
```

Compute the fraction of the sample values in which the hypothesis is not rejected.

```
meanc( (chistat .> 7.56) .and (chistat .< 30.19) );
```

Graph the empirical distribution of the test statistic.

```
{ c4,m4,freq4 } = histp(chistat,30);
```

6.5 Summary Statement

Section 6.5 contains a summary of the contents of Chapter 6. It is a convenient place to update PROC MYOLS written in Chapter 5. Add t-statistics to test the hypotheses that the parameters, individually, are zero.
Your program should look something like the following.

```
proc(2) = myols(x,y);
   local *;

   t = rows(x);
   k=cols(x);
   df = t - k;

   b = y/x;
   ehat = y - x*b;
   sse = ehat'ehat;
   sighat2 = sse/df;

   covb = sighat2*invpd(x'x);
   stderr  =  sqrt(diag(covb));
   tstat  =  b./stderr;

   ybar = meanc(y);
```

```
    sst = y'y - t*(ybar^2);
    r2 = 1 - (sse/sst);
    rbar2 = 1 - (t - 1)*(1 - r2)/df;

    format 8,2;
    "Number of observations: " t;
    "Degrees of freedom:     " df;

    format 10,5;
    "Sum of Squared Errors: " SSE;
    "Total Sum of Squares:  " SST;
    "R-squared:             " R2;
    "R-bar-squared:         " Rbar2;
    "Sigmahat^2:            " sighat2;


    "Standard Error:        " sqrt(sighat2);
     ?;
    "Coeffs   Std. Errs.  T-Stats  P-value ";
     b~stderr~tstat~cdftc(tstat,df);

     ?;
    "Variance-Covariance Matrix for b: ";
     covb;


  retp(b,covb);
endp;
```

6.6 Asymptotic Properties of the Least Squares Estimator

In this Section the asymptotic or large sample distributions of the Least Squares estimator are studied. Under certain conditions the least squares estimator has a normal distribution in large samples no matter what the distribution of the original population.
To examine this phenomenon we will use the Monte Carlo design from Chapter 5.10 which was based on uniform random disturbances with mean zero and variance 2. Create T = 500 Monte Carlo samples and the least squares estimates from Equation 5.10.1 using PROC MC. Recall that uniform (0,1) random numbers have been SAVEed to the files E1UNI.FMT and E2UNI.FMT.

```
t = 20;                              /*  define parameters    */
k = 3;
nsam = 500;
```

```
let beta = 10.0 0.4 0.6;

load x[t,k] = judge.x;              /*  create X             */

load e1  =  e1uni.fmt;              /*  create errors        */
load e2  =  e2uni.fmt;
e1  =  sqrt(2)*e1;
e2  =  sqrt(2)*e2;

{b1,s1} = mc(x,beta,e1);            /*  execute Monte Carlo  */
{b2,s2} = mc(x,beta,e2);

b = b1~b2;                          /*  stack parameters     */
sighat2 = s1'~s2';

b1 = 0; b2 = 0;                     /*  clear                */
s1 = 0; s2 = 0;
```

Compute the means of the parameter estimates and compare them to the results in Section 5.10 to verify that they are the same.

```
meanc(b') meanc(sighat2');
```

The asymptotic distribution result for b in Equation 6.6.21 is made operational by dropping the "limit" from Q and simplifying. The result is that `b` is "approximately" normal in large samples with the usual mean and covariance matrix. Consider the asymptotic distribution of the estimator for β_2. Standardize the random variable by subtracting `beta2` from its estimator and dividing by the true standard error.

```
ixx = invpd(x'x);
z2 = (b[2,.]' - beta[2,1])/(sqrt(2*ixx[2,2]));
```

Calculate the mean and standard deviation of the standardized variable.

```
meanc(z2) stdc(z2);
```

Now the question is, what is the probability distribution of `z2`? If the asymptotic theory "works" the distribution should be N(0,1), if T = 20 is sufficiently large. Write a procedure PROC ZGOF to carry out a simple Chi-square "goodness-of-fit" test, which you studied in your basic statistics course. It compares the observed to expected frequencies. The test statistic has a Chi-square distribution if the observed values come from the distribution used to form the expected frequencies. The hypothesis is rejected if the test statistic is too large. The PROC ZGOF takes as argument the vector of observed values and prints the value of the Chi-square statistic (21 degrees of freedom) and the p-value of the test for a N(0,1) distribution.

```
proc zgof(z);
local *;

nsam = rows(z);
v = seqa(-3,.3,21); v = -5|v|5;    /*  define intervals     */

freq = counts(z,v);                /*  observed freq.       */
freq = freq[2:23,.];

cdfval = cdfn(v);                  /*  calc. expected freq. */
prob = cdfval[2:23,.] - cdfval[1:22,.];
expected = nsam * prob;

/*  test statistic */

gof = sumc(( (freq - expected)^2) ./ expected);
pval = cdfchic(gof,21);            /*  p-value              */

"chi-square statistic : " gof;     /*  print                */
"p-value              : " pval;
retp("");
endp;
```

Place PROC ZGOF in memory and use it to test the distribution of z2, and clear memory.

```
zgof(z2);
b = 0; z2 = 0; sighat2 = 0;
```

As the sample size increases the asymptotic approximation to the distribution should get better. Try the experiment again for T = 40. Construct the larger x by stacking x on top of itself. Larger vectors of errors are obtained by stacking as well.

```
t = 40;
x = x|x;                           /*  construct X          */

e1 = e1[.,1:125]|e1[.,126:250];    /*  construct errors     */
e2 = e2[.,1:125]|e2[.,126:250];

{b1,s1} = mc(x,beta,e1);           /*  run Monte Carlo      */
{b2,s2} = mc(x,beta,e2);

b = b1~b2;                         /*  stack                */
b1=0; b2=0; s1=0; s2=0;            /*  clear                */

ixx = invpd(x'x);
z2 = (b[2,.]'-beta[2,1])/          /*  create z2            */
        sqrt(2*ixx[2,2]);
```

```
zgof(z2);                              /*  carry out test       */
b = 0; z2=0;                           /*  clear                */
```

Repeat for T = 10.

```
t = 10;

x=x[1:10,.];                           /*  use 10 rows of X     */

load e1  =  e1uni.fmt;                 /*  create errors        */
load e2  =  e2uni.fmt;

e1  =  sqrt(2)*e1;
e2  =  sqrt(2)*e2;

e1 = e1[1:10,.]~e1[11:20,.];
e2 = e2[1:10,.]~e2[11:20,.];

{b1,s1} = mc(x,beta,e1);
{b2,s2} = mc(x,beta,e2);               /*  run Monte Carlo      */

b = b1~b2;                             /*  stack                */
b1 = 0; b2 = 0; s1 = 0; s2 = 0;        /*  clear                */

ixx = invpd(x'x);
z2 = (b[2,.]' - beta[2,1])/
     sqrt(2*ixx[2,2]);                 /*  create z2            */

zgof(z2);
b = 0; z2 = 0;                         /*  carry out test       */
```

At this point you may be questioning how big the sample must be before the asymptotic distribution starts to take effect. When the errors are independent and identically distributed uniform random numbers it doesn't take a very large sample. But be assured that in most situations you will encounter "asymptotic normality" does not occur in such small samples. To convince yourself that the distribution is not approximately normal in samples of all sizes, let T = 5.

```
t = 5;
x = x[1:5,.];

load e1  =  e1uni.fmt;
load e2  =  e2uni.fmt;

e1  =  sqrt(2)*e1;
e2  =  sqrt(2)*e2;

e1 = e1[1:5,.]~e1[6:10,.]~e1[11:15,.]~e1[16:20,.];
e2 = e2[1:5,.]~e2[6:10,.]~e2[11:15,.]~e2[16:20,.];
```

```
{b1,s1} = mc(x,beta,e1);
{b2,s2} = mc(x,beta,e2);

b = b1~b2;
b1 = 0; b2 = 0; s1 = 0; s2 = 0;


ixx = invpd(x'x);
z2 = (b[2,.]'-beta[2,1])/
     sqrt(2*ixx[2,2]);

zgof(z2);
```

Chapter 7

Bayesian Inference: II

7.1 Introduction

In this chapter the Bayesian framework of Chapter 4 is extended to the normal linear statistical model. First, a simple model of the consumption function with only one coefficient is analyzed. Second, an example of a normal linear model with more coefficients but a known variance is used. There is a short digression to consider Bayes' point estimation before returning to the standard model with unknown variance. Prior, posterior, and posterior distributions with non-informative priors are examined.

7.2 A Simple Model

LOAD the data from file TABLE7.1. The first column represents consumption (Y) and the second column represents income (X). Check the data.

```
load dat[15,2] = table7.1;
y = dat[.,1];
x = dat[.,2];
format 8,5;
y~x;
```

Compute the least squares estimate of the marginal propensity to consume, β.

```
b = y/x;                                /*  Eq. 7.2.2 */
b;
```

Construct a 95% confidence interval for β, assuming that the variance of the error term is known and equal to 2.25.

```
sigma2 = 2.25;
interval = 1.96*sqrt(sigma2/(x'x));
(b - interval)~(b + interval);          /*  Eq. 7.2.3 */
```

In Section 7.2.1 Bayesian inference with an informative prior is illustrated using the consumption function model. Suppose that there is a 90% probability that beta lies between 0.75 and 0.95, as in Equation 7.2.4. Simultaneously solve the system of equations, below (7.2.5), $\bar{\beta} - 1.645 * \bar{\sigma_\beta} = .75$ and $\bar{\beta} + 1.645 * \bar{\sigma_\beta} = .95$ for the mean, $\bar{\beta}$ and the standard deviation, $\bar{\sigma_\beta}$, of the prior distribution.

```
let a[2,2] = 1 -1.645
             1  1.645;
let c = .75 .95;
p = c/a;
bbar = p[1,1];
sigbar = p[2,1];
bbar sigbar;                              /*  Eq. 7.2.6 */
```

Given the estimates of $\bar{\beta}$ and $\bar{\sigma}$, compute the probability that $\beta > 1$.

```
z = (1-bbar)/sigbar;
cdfnc(z);
```

Compute the probability that $\beta < 0$. (You answer here differs somewhat from text because of rounding error.)

```
z = (0 - bbar)/sigbar;
cdfn(z);
```

Compute the mean and variance of the posterior distribution, where h_o is the precision of the prior information, h_s is the precision of the sample information, and h_1 is the precision of the posterior information. See Equations 7.2.11 and 7.2.12 in the text. Compare your results to those on page 280 of the text.

```
h0 = 1/(sigbar^2);
h0;

hs = (x'x)/sigma2;
hs;

bdb = (hs*b + h0*bbar)/(hs + h0);       /* Eq. 7.2.11 */
bdb;

h1 = h0 + hs;
h1;

sdb2 = 1/h1;                            /* Eq. 7.2.12 */
sdb = sqrt(sdb2);
sdb;
```

Examine the prior and posterior distributions. See Figure 7.1 in the text. First write a function to compute the p.d.f. for the normal distribution using the standard normal p.d.f.

```
fn pdfnorm(mean,std,b) = pdfn( (b-mean)./std )./std;
```

Create a sequence of βs in the relevant range, and compute the prior p.d.f., and graph.

```
bvec = seqa(.6,.005,101);
library qgraph;
pdfprior = pdfnorm(bbar,sigbar,bvec);  /*  Eq. 7.2.9 */
xy(bvec,pdfprior);
```

Compute the posterior p.d.f. and graph it with the prior.

```
pdfpost = pdfnorm(bdb,sdb,bvec); /*  Eq. 7.2.13 and  */
xy(bvec,pdfprior~pdfpost);       /*  Eq. 7.2.15      */
```

Bayesian inference with a noninformative prior is treated in Section 7.2.2. Graph the posterior distribution with a noninformative prior, in addition to the distributions graphed above. (See Figure 7.2 in the text.)

```
varx = sigma2/(x'x);
pdfnon = pdfnorm(b,sqrt(varx),bvec);   /* Eq. 7.2.21 */
xy(bvec,pdfprior~pdfpost~pdfnon);      /* Eq. 7.2.22 */
```

7.3 Bayesian Inference for the General Lineral Model with Known Disturbance Variance

In this Section the posterior distribution for vector of linear model parameters is derived under the assumption that the error variance σ^2 is known. If the prior distribution for the regression parameters is multivariate normal the posterior distribution is also shown to be multivariate normal. If the prior distribution is noninformative the posterior distribution is proportional to the likelihood function and again multivariate normal.

7.4 An Example

To illustrate the use of a natural conjugate prior a Cobb-Douglas production function is hypothesized. See Equations 7.4.1 and 7.4.2.
Prior information is such that $P(.2 < \beta_2 < .8) = P(.2 < \beta_3 < .8) = .9$. Compute the implied mean and variance using the following equations: $\bar{\beta}_2 - 1.645\bar{\sigma}_{\beta_2} = 0.2$ and $\bar{\beta}_2 + 1.645\bar{\sigma}_{\beta_2} = 0.8$. These equations are a representation of Equation 7.4.9.

```
let a[2,2] = 1 -1.645
             1  1.645;
let c = 0.2 0.8;
p = c/a;
```

```
bbar2 = p[1,1];
sbar2 = p[2,1];
sbar22 = sbar2^2;
bbar2 sbar2 sbar22;                    /* See Eq. 7.4.10 */
```

Prior information also implies that $P(-10 < \beta_1 < 20) = .9$. Solve these two equations simultaneously: $\bar{\beta}_1 - 1.645\bar{\sigma}_{\beta_1} = -10$ and $\bar{\beta}_1 + 1.645\bar{\sigma}_{\beta_1} = 20$. These equations come from Equation 7.4.6.

```
let c = -10 20;
p = c/a;
bbar1 = p[1,1];
sbar1 = p[2,1];
sbar11 = sbar1^2;
bbar1 sbar1 sbar11;                    /* See Eq. 7.4.12 */
```

To compute the covariance for β_2 and β_3 use the prior information that $P(.9 < \beta_2+\beta_3 < 1.1) = .9$. First, solve these two equations: $\overline{\beta_2 + \beta_3} - 1.645\bar{\sigma}_{\beta_2+\beta_3} = 0.9$ and $\overline{\beta_2 + \beta_3} + 1.645\bar{\sigma}_{\beta_2+\beta_3} = 1.1$

```
let c = 0.9 1.1;                       /* See Eq. 7.4.4 */
p = c/a;
bbar23 = p[1,1];
sbar23 = p[2,1];
bbar23 sbar23;
```

Use the definition of the variance of a sum to compute the covariance:

```
cov23 = (sbar23*sbar23 - 2*sbar2*sbar2)/2;
cov23;                                 /*  Eq. 7.4.14   */
```

Create the variance-covariance matrix for β, noting that β_2 and β_3 are assumed to have the same prior distribution. See Equation 7.4.15 in the text.

```
vcbar = sbar11~       0~       0
       |0~       sbar22~   cov23
       |0~         cov23~ sbar22;
vcbar;
```

Construct a vector of prior means.

```
bbar = bbar1|bbar2|bbar2;
bbar;
```

Now LOAD the data in file TABLE7.2, define `ypf` to be the vector of observations on the dependent variable and `xpf` to be the matrix of regressor values.

```
load dat[20,4] = table7.2;
ypf = dat[.,1];
xpf = dat[.,2:4];
ypf~xpf;
```

Compute the least squares estimator and its covariance matrix. Assume that σ^2 is known and equal to .09.

```
b = ypf/xpf;
b';                                    /*  Eq. 7.4.16  */

sigma2 = 0.09;
sigma = sqrt(sigma2);
varcov = sigma2*invpd(xpf'xpf);        /*  Eq. 7.4.17  */
```

Compute the mean and covariance matrix for the posterior distribution. See Equations 7.4.18 and 7.4.19 in the text.

```
vcdb = inv( invpd(vcbar) + (xpf'xpf)/sigma2 );
vcdb;
?;
bdb = vcdb* (invpd(vcbar)*bbar  + ((xpf'xpf)/sigma2)*b);
bdb';
```

Graph the marginal prior and posterior parameter distributions. Begin with the prior distribution for β_2, as in Figure 7.3 in the text.

```
bvec = seqa(0,.01,101);
pdfprior = pdfnorm(bbar2,sbar2,bvec);
xy(bvec,pdfprior);
```

Now add the posterior distribution.

```
pdfpost = pdfnorm(bdb[2,1],sqrt(vcdb[2,2]),bvec);
xy(bvec,pdfprior~pdfpost);
```

Repeat the exercise for β_3. Recall that the prior distributions for β_2 and β_3 are identical.

```
pdfpost = pdfnorm(bdb[3,1],sqrt(vcdb[3,3]),bvec);
xy(bvec,pdfprior~pdfpost);
```

Compute the 90% HPD interval for β_2 using the prior and posterior densities.

```
(bbar2 - 1.645*sbar2)~(bbar2 + 1.645*sbar2);
interv = 1.645*sqrt(vcdb[2,2]);
(bdb[2,1] - interv)~(bdb[2,1] + interv);
```

In Chapter 3.6 the relationship between confidence intervals and hypothesis tests is explored. Here a PROC is given that provides a basis for graphing Figure 3.19 in ITPE2. It can be used to plot confidence ellipses for two means, β_2 and $\beta 3$, of a joint normal distribution. PROC `CONFID` takes four arguments: (1) `b`, the estimates of the means; (2) `varcov`, the variance- covariance matrix of the parameter estimates; (3) `fstat`, the relevant test statistic value, which here is the chi-square value divided by 2; and (4) `pos`, a (2 x 1) vector containing the positions of values of β_2 and β_3 in the vector of estimates β. The procedure returns a matrix with two columns–the values to be used in plotting the confidence ellipse. Store the PROC for later use, perhaps in the SRC subdirectory as file confid.g so that it can be automatically "loaded" when called. See your GAUSS manual about automatic loading of procedures.

```
proc CONFID(b,varcov,fstat,pos);
   local *;

   b1 = b[pos[1,1],1];
   b2 = b[pos[2,1],1];
   R = zeros(2,rows(b));
   R[1,pos[1,1]] = 1;
   R[2,pos[2,1]] = 1;

   A = inv(R*varcov*R');

   q = a[1,1]*a[2,2] - a[1,2]*a[1,2];
   lb = b2 - sqrt(2*fstat*a[1,1]/q);
   ub = b2 + sqrt(2*fstat*a[1,1]/q);
   beta2 = seqa(lb,(ub-lb)/100,101);

   csq =  (b2 - beta2)^2*( - q/a[1,1]^2)
          + fstat*2/a[1,1];

   c = sqrt(abs(csq));

   beta1a = b1 + (b2-beta2)*a[1,2]/a[1,1] + c;
   beta1b = b1 + (b2-beta2)*a[1,2]/a[1,1] - c;


 retp((beta2|rev(beta2))~(beta1a|rev(beta1b)));
endp;
\
```

Compute and graph the joint 95% HPD region for β_2 and β_3 and again compare the prior with the posterior results. You will need to load the procedure PROC `CONFID` into memory. Or it must be in file that can be found by **GAUSS**'S autoloading feature. The procedure takes four arguments. The first is the

entire vector of means of the coefficients. The second argument is the variance-covariance matrix, the third the statistic value, which in this case is $\chi^2/2$. Last is a vector containing the positions of the parameters to use, in this case 2 and 3. Begin with the confidence region from he prior distribution. Compare to Figure 7.4.

```
let pos = 2 3;
d = confid(bbar,vcbar,5.99/2,pos);
xy(d[.,1],d[.,2]);
```

Now add the confidence region for the posterior distribution.

```
d1 = confid(bdb,vcdb,5.99/2,pos);
d = d~d1;
xy(d[.,1 3],d[.,2 4]);
```

7.5 Point Estimation

Simulate data to compute the empirical Bayes' estimation. (See Section 7.5.2 of the text.) First create a (20 x 4) matrix of normally distributed error terms with a variance equal to 2.

```
k = 4;
n = 20;
ssq = 2;
e = sqrt(ssq)*rndn(n,k);
```

Assume that the (K x 1) vector of βs has a mean of μ and variance τ^2. Create a vector `beta`. See Equation 7.5.19 in the text.

```
tau2 = 4;
mu = 2;
beta = mu + sqrt(tau2)*rndn(4,1);
beta';
```

Use the vector `beta` and matrix `e` to create a matrix of observations, `y`.

```
y = beta' + e;
```

The least squares estimator is the vector of sample means. See Equation 7.5.12.

```
ybar = meanc(y);
ybar';
```

How do these values compare with the true βs?
The estimate for μ is the sample mean of $\bar{y}$. See the discussion following Equation 7.5.21 in the text.

```
ydb = meanc(ybar);
ydb;
```

How does `ydb` compare with the true value of $\mu = 2$?
An estimate of the weight attached to μ when computing the posterior mean is:
(See Equation 7.5.24 in the text.)

```
wt = ((k-3)*sigma2/n)/(sumc((ybar - ydb)^2));
wt;
```

The empirical Bayes' estimator is computed from Equation 7.5.25 in the text.

```
bdb =  wt*ydb + (1-wt)*ybar;
bdb';
```

7.6 Comparing Hypotheses and Posterior Odds

Here we continue the production function example. Make sure `YPF, XPF, B, BBAR, VCBAR`, and `SIGMA2` from Section 7.4 above are in memory.

```
ypf; xpf; b; bbar; vcbar; sigma2;
```

From the discussion following Equation 7.6.10 in the text, compute the matrix `a`.

```
a = invpd(vcbar/sigma2);
```

Compute `rss1` and `q1` from Equations 7.6.13 and 7.6.14.

```
rss1 = (ypf - xpf*b)'(ypf - xpf*b);

q1 = (b-bbar)'(inv(inv(A)+inv(xpf'xpf)))*(b-bbar);
```

Define `q` and `z` using Equation 7.6.17.

```
q = ypf - xpf[.,3];
z = xpf[.,1]~(xpf[.,2] - xpf[.,3]);
```

Define `gbar`, `ghat`, and `bb` (B in the text). See the discussion following Equation 7.6.18 and before 7.6.21.

```
gbar = bbar[1 2,.];
bb = invpd(vcbar[1 2,1 2]/sigma2);
ghat = z'q/z'z;
```

From Equations 7.6.21 and 7.6.22, compute `rss0` and `q0`.

```
rss0 = (q - z*ghat)'(q - z*ghat);
q0 = (ghat-gbar)'(invpd(invpd(bb)+invpd(z'z)))*
                                        (ghat-gbar);
```

Last, using Equation 7.6.24 compute the posterior odds ratio given a prior odds ratio of unity.

```
t1 = sqrt( det(bb)/(det(bb + z'z)) );
t2 = sqrt( det(a)/ (det(a + xpf'xpf)) );
k01 = (t1/t2)*exp(-(rss0-rss1+q0-q1)/(2*sigma2));
k01;
```

Compute the chi-square statistic which would be calculated if the same test were performed within a sampling theory framework. (See Equation 7.6.27.)

```
chi = (rss0 - rss1)/sigma2;
chi cdfchic(chi,1);
```

7.7 Bayesian Inference for the General Linear Model with Unknown Disturbance Variance

Continue the production function example from Section 7.5, now assuming that the disturbance variance is unknown.
As in Section 4.4 (page 143), solve for the parameters of the inverted gamma, on the assumption that cdfchic(s/.09,v) = .5 and cdfchic(s/(.65^2),v) = .95. Do so by creating a matrix of possible values of s and v and finding the pair that comes closest to simultaneously satisfying the two equations.

```
v = seqa(1,1,10);
ssq = seqa(.05,.0001,300);
r = v.*ssq';
q = abs( cdfchic(r/.09,v) - .5 ) +
    abs(cdfchic(r/(.65^2),v) - .95);
i = minindc(minc(q));
j = minindc(minc(q'));
vbar = v[j,1];
sbarsq = ssq[i,1];
vbar sbarsq;
sbar = sqrt(sbarsq);
```

Compute the mean and mode for the prior density for σ using Equations 4.4.8, on page 142.

```
mean = sqrt(vbar/2)*gamma((vbar-1)/2)*sbar/gamma(vbar/2);
mode = sqrt( vbar/(vbar + 1) )*sbar;
mean mode;
```

Compute the mean of the posterior distribution. See Equation 7.7.10.

```
adb = inv(a + xpf'xpf);
bdb = adb*(a*bbar + xpf'xpf*b);
```

Using Equation 7.7.13 in the text, compute the posterior parameter `vdb`.

```
T = rows(ypf);
vdb = T + vbar;
```

Compute the posterior parameter `sdb2` using `vdb` and Equation 7.7.11.

```
sdb2 =
(vbar*sbarsq+ypf'ypf+bbar'A*bbar-bdb'*
                                    (A+xpf'xpf)*bdb)/vdb;
```

Graph the marginal prior and posterior distributions for σ^2 using the inverted gamma p.d.f., given in Equation 7.7.4. In addition, graph the posterior distribution assuming a noninformative prior. Begin by writing a function to compute the p.d.f of the inverted gamma.

```
fn igamma(sigma,v,s) =
     2/(gamma(v/2)) * (v*s*s/2)^(v/2) *
     (1./sigma^(v+1)) .* exp(-v*s*s./(2*sigma.*sigma));
```

Create a vector of σs in the relevant range, compute the p.d.f. for the marginal prior, and graph. See Figure 7.7.

```
sigma = seqa(.001,.01,80);
priors = igamma(sigma,vbar,sbar);
xy(sigma,priors);
```

Now add the marginal posterior distribution for σ.

```
posts = igamma(sigma,vdb,sqrt(sdb2));
xy(sigma,priors~posts);
```

Next compute some parameters needed for the posterior distribution with a noninformative prior (`nonis`), compute it, and add it to the graph. See Section 7.7.4 and the definitions below Equation 7.7.34.

```
k = rows(b);
ssq = (ypf - xpf*b)'(ypf - xpf*b)/(t - k);
nonis = igamma(sigma,t-k,sqrt(ssq));
xy(sigma,priors~posts~nonis);
```

Graph the prior and posterior distributions for β_2. Consider both an informative and non-informative prior. See Equations 7.7.26 and 7.7.27. First write the probability density function for the t-distribution: μ is the mean, h is the precision, v is the degrees of freedom and x is the value of the random variable. The formula comes from Equation 7.7.19 with p = 1.

```
fn pdft(u,h,v,x) =
   gamma((v+1)/2)*(1/(gamma(.5)*gamma(v/2)))*sqrt(h/v)
   * (1 + ( h*(x-u)^2)/v)^(-(v+1)/2);
```

Specify a vector of b's in the relevant range and graph the prior distribution.

```
ainv = inv(a);
bvec = seqa(0,.01,100);
priorb = pdft(bbar[2,.],1/(sbarsq*ainv[2,2]),vbar,bvec);
xy(bvec,priorb);
```

Now add the posterior distribution to the graph.

```
postb = pdft(bdb[2,.],1/(sdb2*adb[2,2]),vdb,bvec);
xy(bvec,priorb~postb);
```

Compute the parameters for the posterior with a non-informative prior, and include it in the graph. See Equation 7.7.37.

```
ixx = invpd(xpf'xpf);
nonib = pdft(b[2,1],1/(ssq*ixx[2,2]),T-k,bvec);
xy(bvec,priorb~postb~nonib);
```

Do the same for β_3.

```
priorb = pdft(bbar[3,1],1/(sbarsq*ainv[3,3]),vbar,bvec)
xy(bvec,priorb);

postb = pdft(bdb[3,1],1/(sdb2*adb[3,3]),vdb,bvec)
xy(bvec,priorb~postb);

nonib = pdft(b[3,1],1/(ssq*ixx[3,3]),T-k,bvec);
xy(bvec,priorb~postb~nonib);
```

Compute and graph confidence intervals for β_2 and β_3 using the informative prior distribution, as in Figure 7.8 in the text.

```
vcbar = sbarsq*ainv;
vcdb = sdb2*adb;
let pos = 2 3;
d = confid(bbar,vcbar,6.94,pos);
xy(d[.,1],d[.,2]);
```

Now add the confidence region using the posterior distribution.

```
d1 = confid(bdb,vcdb,3.40,pos);
d = d~d1;
xy(d[.,1 3],d[.,2 4]);
```

Can you add the confidence region using the posterior with a non-informative prior to the graph? (Hint: F(2,17) = 3.59, the variance-covariance matrix is `ssq*ixx`, and the parameter estimates are in `b`.)

Chapter 8

General Linear Statistical Model

In this chapter the classical linear regression model is generalized by considering situations where the error covariance matrix is not equal to a scalar times an identity matirx, i.e., $Cov(e) \neq \sigma^2 I_T$

8.1 The Statistical Model and Estimators

In this chapter you will create a data set in which the error terms are first- order autoregressive. The parameters are then estimated using ordinary least squares and generalized least squares. A Monte Carlo study allows you to carefully compare the characteristics of the two procedures and to see the pitfalls in using ordinary least squares inappropriately.
The Monte Carlo experiment will be set up as in Section 8.1.5 in ITPE2. The design matrix X and the true parameter vector β are as in Chapter 6.

```
t = 20;
k = 3;
load x[t,k] = judge.x;
let beta = 10.0 0.4 0.6;
```

Set the parameter $\rho = 0.9$ and $\sigma^2 = .0625$.

```
rho = 0.9;
sigma2 = .0625;
```

Create the underlying covariance matrix for the error terms, following Equation 8.8.21 in the text. While there are no doubt more clever ways to proceed, define Ψ to be a (T x T) identity matrix and then simply fill in the off-diagonal elements with powers of ρ, using nested DO-LOOPS, and taking advantage of the symmetry of Ψ.

```
psi = eye(t);                  /*  (T x T) identity     */

i = 1;                         /*  begin "row" loop     */
do while i le t;

j = i + 1;                     /*  begin "col" loop     */
do while j le t;

psi[i,j] = rho^(j - i);        /*  i,j-th element       */
psi[j,i] = psi[i,j];           /*  j,i-th element       */

j = j+1;
endo;                          /*  end column loop      */

i = i+1;
endo;

psi = psi ./ (1 - rho^2);      /*  See Eq. 8.1.21       */
```

With X given and knowledge of Ψ and σ^2 you can compute the true covariance matrices of both the ordinary least squares and generalized least squares estimators of the parameters. For the ordinary least squares covariance use Equation 8.1.24 in the text:

```
ixx = invpd(x'x);
covb = sigma2*ixx*x'*psi*x*ixx;
covb;
```

The covariance of the generalized least squares estimator is computed by Equation 8.1.23 in the text.

```
ipsi = invpd(psi);
covbhat = sigma2*invpd(x'*ipsi*x);
covbhat;
```

Note that the variances (shown on the diagonals of the covariance matrices) of the generalized least squares estimator are smaller than those of the OLS estimator.
You can also compute the expected value of the usual estimator of the error variance, given your special knowledge of the true σ^2, as in Equation 8.1.25 of ITPE2.

```
tr1 = sumc(diag(psi));
tr2 = sumc(diag(x'*psi*x*ixx));
esighat2 = sigma2*(tr1 - tr2)/(t - k);
esighat2;
```

How does the expected value of the biased estimator of the variance compare with the true value of $\sigma^2 = 0.0625$?
The most complicated part of simulating the data generation process is creating the error terms. Make use of the fact that the autocorrelated error, $e_t = e_{t-1} + v_t$, can be created using a normally distributed error term, `estar`, with mean 0 and variance σ^2. First create a vector `estar` using the first 20 "official" normal random numbers, which have a N(0,1) distribution.

```
open f1 = nrandom.dat;        /*  open file          */
estar = readr(f1,t);          /*  read T obs.        */
f1 = close(f1);               /*  close file         */
estar = sqrt(sigma2)*estar;   /*  change variance    */
```

Next use the transformation outlined below Equation 8.1.21 in the text, and apply it to `e`. The first element in `e` is constructed using a special formula. The remainder can be constructed iteratively using a first order autocorrelation process.

```
e = zeros(t,1);                  /* initialize vector */
e[1,1] =                         /*  create element 1 */
    estar[1,1]/sqrt( 1 - (rho^2));

i = 2;                           /*  begin loop       */
do while i le t;
e[i,1] = rho*e[i-1,1] + estar[i,1]; /*  e(i)          */
i = i+1;
endo;                            /*  end loop         */

e';                              /*  print            */
```

Given e[1,1] the function RECSERAR, which creates an autoregressive recursive series, could also be used.

```
e = recserar(estar,e[1,1],rho);
e';
```

Now create the vector `y`.

```
y = x*beta + e;
```

Compute least squares estimates, b and $\hat{\sigma}^2$.

```
b = y/x;
b';
ehat = y - x*b;
sighat2 = ehat'ehat/(t - k);
sighat2;
```

Use the usual (and in this case, incorrect) formula to compute the covariance matrix for b. Compare it to the true covariance, $E[(b - E(b))(b - E(b))']$.

```
badcovb = sighat2*ixx;
badcovb covb;
```

Compute generalized least squares estimates, $\hat{\beta}$ and $\hat{\sigma}_g^2$.

```
ipsi = invpd(psi);
bhat = invpd(x'ipsi*x)*(x'ipsi*y);
eghat = y - x*bhat;
sighatg2 = eghat'ipsi*eghat/(t - k);
sighatg2;
```

As with LS estimation the GLS estimates could be obtained efficiently using **GAUSS** division, " / ".

```
bhat = (x'ipsi*y)/(x'ipsi*x);
bhat;
```

In the Monte Carlo experiment below, it will be useful to know that an alternative way to compute $\hat{\sigma}_g^2$ is:

```
sighatg2 = sumc( (ipsi*eghat) .* eghat)/(t - k);
sighatg2;
```

Now compute the covariance matrix for the generalized least squares coefficient estimates, and compare with the true values in `covbhat`.

```
ecovbhat = sighatg2*invpd(x'*ipsi*x);
ecovbhat covbhat;
```

Compare your estimates to correspondents in the first row of Table 8.1 in ITPE2. They should be the same.
In order to carry out a Monte Carlo study of generalized least squares first write a procedure to simulate NSAM data sets at one time, compute the least squares and generalized least squares estimates. The alternative is to use many DO-LOOPS, which would take considerably longer to execute. PROC `MCG` takes as arguments `x`, `beta`, `rho`, and a (T x NSAM) matrix of N(0,σ^2) random disturbances. It returns all the estimates stacked into one matrix. It is long so place it in a separate file, and then run it.

```
proc  MCG(x,beta,rho,estar);
   local t,k,nsam,psi,i,e1,y,b,sighat2,ipsi,bhat,sighatg2;

   t = rows(x);               /*  define constants   */
   k = cols(x);
   nsam = cols(estar);
```

```
    psi = eye(t);              /*  create psi         */
    i = 1;
    do while i le t;
    j = i+1;
    do while j le t;
    psi[i,j] = rho^(j - i);
    psi[j,i] = psi[i,j];
    j = j+1;
    endo;
    i = i+1;
    endo;
    psi = psi ./ (1-rho^2);

    e1 = estar[1,.]/sqrt(1 - (rho^2));   /*  create e */
    y = x*beta                           /*  create y */
      + recserar(estar,e1,rho*ones(1,nsam));

    b = y/x;                             /*  ols      */
    sighat2 = sumc((y-x*b).*(y-x*b))/(T-k);

    ipsi = invpd(psi);                   /*  gls      */
    bhat = (x'ipsi*y)/(x'ipsi*x);
    sighatg2 = sumc( (ipsi*(y - x*bhat)) .*
                                (y - x*bhat))/(t - k);

    retp(b|sighat2'|bhat|sighatg2');
endp;
```

Use the procedure to compute estimates for 250 samples. Create the matrix `estar`.

```
load estar = e1nor.fmt;
estar = sqrt(sigma2)*estar;
est = MCG(x,beta,rho,estar);
```

Print out to the screen the estimates of the first ten samples, where each row represents the estimates for one sample. Compare these estimates to the values in Table 8.1.

```
format 6,4;
est[.,1:10]';
```

Add another 250 samples and stack them next to the estimates from the first 250 samples. Then clear `estar` from memory.

```
load estar = e2nor.fmt;
```

```
estar = sqrt(sigma2)*estar;
est = est~MCG(x,beta,rho,estar);
estar = 0;
```

Calculate the mean values of the parameter estimates from the 500 samples and compare them to the true values.

```
meanc(est')';
```

Compute the true variances of the estimated coefficients using covb and covbhat computed above. Compare them with the sampling variances from the Monte Carlo study. Note, the **GAUSS** function STDC uses divisor NSAM - 1, and thus we deflate the estimated variances to make them strictly comparable to the diagonal elements of Equation 8.1.26.

```
var = diag(covb)|diag(covbhat);
var';
b = est[1:3 5:7,.];
(stdc(b')^2)'*(499/500);
```

Now compute the estimated standard errors using the ordinary least squares formula.

```
badse = sqrt( est[4,.].*diag(ixx) );
```

Do the same for the generalized least squares estimated coefficients.

```
bhatse = sqrt( est[8,.].*diag(invpd(x'ipsi*x)) );
```

Compare mean estimated standard deviations to truth.

```
sqrt(var)';
meanc(badse')';; meanc(bhatse')';
```

8.2 The Normal Linear Statistical Model

In this Section it is shown that if the random vector of disturbances has a multivariate normal distribution then the GLS estimator is the same as the maximum likelihood estimator. The ML estimator of σ^2 has the divisor T rather than T - K.

8.3 Sampling distributions of the Maximum Likelihood Estimators

If Ψ is known, the ML estimator of β has a normal distribution with the usual mean and covariance. Likewise the unbiased estimator of σ^2 has a multiple of a chi-square distribution with (T - K) degrees of freedom.

8.4 Interval Estimators

Given the results in Sections 8.2 and 8.3 it is not surprising that all usual estimation procdures may be applied, with substitution of the GLS estimators and the appropriate covariance matrix.

8.5 Hypothesis Testing

Linear hypotheses may be tested in the usual way, again with substitutions of proper estimators and covariance matrices. Continue the Monte Carlo experiment.
Perform t-tests at the 5% level of significance for each coefficient for each sample. (See Section 8.5 of the text.) Remember that the computed t-statistic for the ordinary least squares estimates do not in fact have a t-distribution.

```
tstat = (est[1:3 5:7,.] - (beta|beta))./(badse|bhatse);
meanc( (abs(tstat) .>= 2.11)')';
```

How close are the percentages of times the hypotheses are rejected to the true probability of 5%?
Graph the distribution of the least squares estimate of β_3.

```
library qgraph;
{ c1,m1,freq1 } = histp(est[3,.]',30);
```

Graph the distribution of the generalized least squares estimate of β_3.

```
{ c2,m2,freq2 } = histp(est[7,.]',30);
```

Graph the distribution of the least squares t-statistics.

```
{ c3,m3,freq3 } = histp(tstat[3,.]',30);
```

Graph the distribution of the generalized least squares t-statistics.

```
{ c4,m4,freq4 } = histp(tstat[6,.]',30);
```

You can do the same for other parameters.

8.6 The Consequences of Using Least Squares

This section summarizes the material in Chapter 8 regarding the consequences of using Least Squares estimation rules when Generalized Least Squares is appropriate.

8.7 Prediction

In the context of the Generalized Least Squares model it is sometimes possible to improve predictions by taking into account relationships, if any, between current and future observations. The algebra is presented here in some detail and will be applied in Chapter 9.5.5.

Chapter 9

General Linear Model with Unknown Covariance

9.1 Background

In this chapter the general linear model is examined and problems associated an unknown error covariance matrix are treated. The problems of heteroskedasticity and autocorrelation are used as examples.

9.2 Estimated Generalized Least Squares

When the error covariance matrix is not known it must be consistently estimated before the generalized least squares estimator can be used. The resulting estimator is called the Estimated Generalized Least Squares (EGLS) estimator. In this section the asymptotic properties of this estimator are considered and algebraic conditions stated under which those properties hold.

9.3 Heteroskedasticity

An example of a heteroskedastic model is given in Section 9.3.7 in ITPE2. In this example the error term is normal and independently distributed with mean zero, but the error variance is not constant. Instead the error variance follows the model of multiplicative heteroskedasticty described in Section 9.3.4
The data contained in Table 9.1 is provided on the disk that accompanies this book and is contained in the file TABLE9.1. LOAD that data and create the design matrix X.

```
load dat[20,5] = table9.1;
format 10,7;
x = ones(20,1)~dat[.,2 3];
```

Create the vector `sigma2` following (9.3.57) and compare to the tabled values.

```
sigma2 = exp( -3 + 0.3 .* x[.,2] );
sigma2~dat[.,4];
```

Create the vector y using the given parameter values for β and the first 20 official normal random numbers and compare to the tabled values.

```
t = rows(x);                          /*  define t    */
k = cols(x);                          /*  define k    */
let beta = 10 1 1;                    /*  true beta   */

open f1=nrandom.dat;                  /*  open file   */
e = readr(f1,t);                      /*  read t obs. */
f1 = close(f1);                       /*  close file  */

e = sqrt(sigma2) .* e;                /*  create e    */
y = x*beta + e;                       /*  create y    */
y~dat[.,1];                           /*  print y     */
```

Begin by computing the least squares estimates of the coefficients.

```
b = y/x;
format 8,5; b';                       /*  Eq. 9.3.58  */
```

Assuming (incorrectly) that the disturbances are homoskedastic, estimate the covariance matrix of the estimated coefficients.

```
ehat = y - x*b;
sighat2 = ehat'ehat/(t-k);
sighat2;

ixx = invpd(x'x);
badcovb = sighat2*ixx;
badcovb;                               /*  Eq. 9.3.59 */
```

Print out the estimated coefficients in a row, with their (incorrectly) estimated standard errors beneath them.

```
b';                                    /*  Eq. 9.3.60 */
sqrt(diag(badcovb))';
```

Now compute the true covariance matrix for b, following Equation 9.3.61 in the text. You can do this because you have the unusual information of the exact structure of the errors, described in Equation 9.3.57 of the text. The function DIAGRV puts the values of `sigma2` on the diagonal of an identity matrix.

```
phi = diagrv(eye(t),sigma2);
covb = ixx*(x'*phi*x)*ixx;
covb;                                   /*  Eq. 9.3.61 */
```

Compare the incorrectly computed standard errors with the true values.

```
sqrt(diag(badcovb))';
sqrt(diag(covb))';
```

Because you know the covariance matrix of the disturbances it is possible to compute the generalized least squares estimates and their covariance matrix.

```
covg = invpd(x'invpd(phi)*x);
covg;                                  /*  Eq. 9.3.62  */

bg = covg*x'invpd(phi)*y;
bg';                                   /*  Eq. 9.3.63  */
sqrt(diag(covg))';
```

In general you will not know the exact structure of the error covariance matrix and while you may suspect heteroskedasticity is present it is usually necessary to test for its presence. First use the Breusch-Pagan test.
Using `ehat` computed above, compute the dependent variable to be used in the test regression.

```
sigtilde = ehat'ehat/t;
e2 = (ehat^2)./sigtilde;
```

The independent variables to be used for the test are the first two columns of the matrix X.

```
z = x[.,1 2];
```

The estimated coefficients are put into `ahat`.

```
ahat = e2/z;
```

Next compute the total and error sum of squares for the test regression

```
sst = (e2 - meanc(e2))'(e2 - meanc(e2));
e2hat = e2 - z*ahat;
sse = e2hat'e2hat;
```

The Breusch-Pagan test statistic is equal to one-half of the regression sum of squares.

```
q = .5*(sst - sse);
q;
cdfchic(q,1);
```

The statistic `q` is asymptotically distributed as Chi-square with 1 degree of freedom. If `q` is greater than 3.84 (the 5% critical value) it is significant at the 5% level, and the hypothesis of homoskedasticity is rejected.
The Goldfeld-Quandt test requires running two separate regression on subsamples of the data, and computing the residual sum of squares from each regression. In general the data must be sorted according to an increasing error variance before the partitioning. Here, however, if we assume that the error variance increases with the magnitude of the second regressor, the data is already in the proper order.
Include the first eight observations in the first regression.

```
y1 = y[1:8,.];
x1 = x[1:8,.];
b1 = y1/x1;
e1 = y1 - x1*b1;
sse1 = e1'e1;
```

The second regression includes the last eight observations.

```
y2 = y[13:20,.];
x2 = x[13:20,.];
b2 = y2/x2;
e2 = y2 - x2*b2;
sse2 = e2'e2;
```

The F-statistic for the Goldfeld-Quandt test is the ratio of the two residual sum of squares.

```
f = sse2/sse1;
f;
cdffc(f,5,5);
```

At what significance level can you reject the hypothesis of homoskedasticity?

9.3.1 The Estimated Generalized Least Squares Estimator

Assuming the that least squares estimate, `b`, and the residuals, `ehat`, are still in memory, the next step is to estimate the vector α which is used to compute the variances of the disturbances. From Equation 9.3.41 in the text, regress the log of the squared errors on the first two columns of X.

```
q = ln(ehat^2);
z = x[.,1 2];
ahat = q/z;
ahat';                                  /*  Eq. 9.3.64 */
```

Adjust the constant term for bias. Although this is not necessary for the estimates of the coefficients, it will affect the estimated covariance matrix.

```
ahat[1,1] = ahat[1,1] + 1.2704;
ahat';
```

Compute the estimated generalized least squares estimator, following Equation 9.3.65 in the text.

```
psihatv = exp(z[.,2]*ahat[2,1]);   /* diag. elements  */
psihat = diagrv(eye(20),psihatv);  /* diagonal matrix */
invpsi = invpd(psihat);            /* inverse         */
begls = (x'invpsi*y)/(x'invpsi*x); /* est. gls        */
begls';                            /* Eq. 9.3.65      */
```

These parameters could alternatively be estimated by using ordinary least squares on transformed data (also called weighted least squares)

```
ystar = y ./ sqrt(psihatv);
xstar = x ./ sqrt(psihatv);
begls = ystar/xstar;
begls';
```

Estimate paramater σ^2 and the covariance matrix for b.

```
ehatg = y - x*begls;
sighatg2 = ehatg'invpsi*ehatg/(t-k);
covegls = sighatg2*invpd(x'invpsi*x);
covegls;                                /*  Eq. 9.3.66 */
```

Summarize your results, with standard errors reported underneath the coefficients.

```
begls';
sqrt(diag(covegls))';                   /*  Eq. 9.3.67 */
```

9.4 Exercises on Heteroskedasticity

The numercial exercises in this section can be carried out using the skills you have learned in Section 9.3.

9.5 Autocorrelation

In this section of the text the problem of autocorrelation is defined. Procedures for implementing EGLS are presented and tests for autocorrelation given. Begin by considering the Example in Section 9.5.3c of ITPE2.
Type in the data used in Section 9.5.3c in the text.

```
let y = 4  7  7.5  4   2
        3  5  4.5 7.5  5;

let x1 = 2  4  6  3  1
         2  3  4  8  6;

t = rows(y);
x = ones(t,1)~x1;
k = cols(x);
```

Compute the least squares parameter estimates and the least squares residuals.

```
b = y/x;
ehat = y - x*b;
```

Assuming a first-order autoregressive process, compute the least squares estimate of rho given in Equation 9.5.40.

```
et = ehat[2:10,1];
el = ehat[1:9,1];
rhohat =  et/el;
rhohat;
```

An asymptotic test for first-order autoregressive errors is described in Section 9.5.3a in the Text.
The test statistic z has an approximate standard normal distribution under the null hypothesis. The null hypothesis is rejected at the 5% level if the absolute value of Z is greater than 1.96.

```
z = sqrt(t)*rhohat;
z;
```

The Durbin-Watson test is described in Section 9.5.3b of the text. Compute the Durbin-Watson statistic using Equation 9.5.45.

```
d = (et-el)'(et-el)/(ehat'ehat);
d;
```

To compute the exact critical value, use the procedure EXACTDW which is given in section 9.6 begining on page 82. The theory behind this procedure is beyond the scope of this course, so don't worry about "how" it works for now. It takes three arguments: `d`, the Durbin-Watson statistic; `x`, the matrix of explanatory variables; and `rho`, the hypothesized value of ρ. To compute the probability that `d` is less than 1.037 given $\rho = 0$, enter the following:

```
EXACTDW(d,x,0);
```

What is the probability that DW is less than 1.037 assuming that $\rho = .5$?
If you did not have a procedure such as `EXACTDW`, it might be useful to compute Durbin and Watson's (1971) approximation of the critical value. To do so, first create the matrix A, shown in Equation 9.5.46 in the text. Do this by first creating a matrix A_1 which has 1's on the off-diagonal.

```
a1 = zeros(t-1,1)~eye(t-1)|zeros(1,t);
```

Next create a matrix with 2's on the diagonal, and using the matrix A_1, put -1's on the off-diagonals. Then set the two corner elements equal to 1.

```
a = eye(t)*2 - (a1 + a1');
a[1,1] = 1;
a[t,t] = 1;
format 2,0; a;
```

Notice that you can use the matrix A to compute the Durbin-Watson statistic, as shown in Equation 9.5.43 in the text.

```
d = (ehat'a*ehat)/(ehat'ehat);
format 8,4;  d;
```

Following Equations 9.5.60 and 9.5.61 compute the values P and Q.

```
ixx = invpd(x'x);
p1 = sumc(diag(x'a*x*ixx));
p = 2*(t-1) - p1;
format 10,7;  p;

q1 = sumc(diag( x'a*a*x*ixx ));
q2 = sumc(diag( (x'a*x*ixx)*(x'a*x*ixx) ));
q = 2*(3*t-4) - 2*q1 + q2;
q;
```

Now compute the expected value and variance of d. (See Equations 9.5.58 and 9.5.59.)

```
exd = p/(t-k);
exd;
vard = 2*(q - p*exd)/( (t-k)*(t-k+2) );
vard;
```

Using the values of EXDU and VARDU from the tables at the end of the text, compute the parameters `aa` (a in the text) and `bb` (b in the text).

```
exdu = 2.238;
vardu = 0.29824;
bb = sqrt(vard/vardu);
bb;
aa = exd - (exdu*bb);
aa;
```

The approximate critical value is d_u^*.

```
du = 1.32;
dstar = aa + bb*du;
dstar;
```

Compare the value of the critical value with the test statistic D. Do you accept of reject the null hypothesis of no positive autocorrelation?
In Section 9.5.6 of ITPE2 a second example relating to autocorrelation is given. First, load the data given in the file TABLE9.2 on the disk accompanying this book and examine it.

```
load dat[20,3] =  table9.2;
y = dat[.,1];
x = ones(20,1)~dat[.,2 3];
dat=y~x;
dat;
```

Before analyzing this data verify your understanding of the data generation process by generating the vector y. The true parameter values are given on page 411 of ITPE2 and refer to equation (9.6.2)

```
let beta = 10 1 1;
sigma2 = 6.4;
rho = .8;
```

Read in the N(0,1) random disturbances `v` and create the random errors `e` using the "inverse" of the data transformation described in equations (9.5.31) and (9.5.32).

```
t = rows(x);                          /*  define t        */
k = cols(x);                          /*  define k        */

open f1 = nrandom.dat;                /*  open file       */
v = readr(f1,t);                      /*  read t obs.     */
f1 = close(f1);                       /*  close file      */

v = sqrt(sigma2) .* v;                /*  adjust variance */
e = zeros(t,1);                       /*  create e        */
e[1,1] = v[1,1] ./ sqrt(1-rho^2);     /*  first element   */

ind = 2;                              /*  begin loop      */
do while ind le t;
e[ind,1] = rho*e[ind-1,1] + v[ind,1]; /* t'th element   */
ind = ind + 1;                        /* increment index  */
endo;                                 /*  end loop        */
```

Create the vector y and compare it to the data in Table 9.2.

```
y = x*beta + e;
y~dat[.,1];
```

Compute the least squares estimates.

```
b = y/x;
b';

ehat = y - x*b;
t = rows(x);
k = cols(x);
sighat2 = ehat'ehat/(t-k);
sighat2;
```

Compute the least squares estimated covariance matrix for b.

```
badcovb = sighat2*invpd(x'x);
```

Compute the Durbin-Watson statistic to test for autocorrelation.

```
edif = ehat[2:20,1] - ehat[1:19,1];
d = (edif'edif)/(ehat'ehat);
d;
```

Compute the critical value of the Durbin-Watson statistic.

```
EXACTDW(d,x,0);
```

Compute the least squares estimate of ρ.

```
rhohat = ehat[2:20,1]/ehat[1:19,1];
rhohat;
```

Transform the data using the estimated `rho` and the matrix `dat` containing both Y and X.

```
dstar = dat - rhohat*dat[1 1:19,.];
dstar[1,.] = sqrt(1-rhohat^2)*dat[1,.];
```

Take `ystar` and `xstar` out of the matrix `dstar`.

```
ystar = dstar[.,1];
xstar = dstar[.,2:4];
```

Compute the estimated generalized least squares estimator using the transformed data.

```
bhat = ystar/xstar;
bhat';
```

Estimate the covariance matrix using the transformed data.

```
estar = ystar - xstar*bhat;
sighatg2 = estar'estar/(t-k);
sighatg2;

covbhat = sighatg2*invpd(xstar'xstar);
covbhat;
```

Summarize the results with the estimated standard errors. Compare the estimated generalized least squares results with the least squares estimates.

```
bhat';
sqrt(diag(covbhat))';
?;
b';
sqrt(diag(badcovb))';
```

Re-estimate the coefficients, this time adjusting ρ following Equation 9.5.42 in the text. Set the parameter M in that equation equal to 1.

```
arho = rhohat + (2 + 4*rhohat)/t;
arho;
```

Compute the new estimated generalized least squares estimates.

```
dstar = dat - arho*dat[1 1:19,.];
dstar[1,.] = sqrt(1-arho^2)*dat[1,.];
ystar = dstar[.,1];
xstar = dstar[.,2:4];
bhata = ystar/xstar;
bhata';
```

Compute the covariance matrix.

```
estara = ystar - xstar*bhata;
sighata = estara'estara/(t-k);
sighata;

covbhata = sighata*invpd(xstar'xstar);
covbhata;
```

Compare the two generalized least squares estimates.

```
bhat';
sqrt(diag(covbhat))';
?;
bhata';
sqrt(diag(covbhata))';
```

Predict the value of Y_t for $x_{2t} = 20$ and $x_{3t} = 20$, ignoring the autocorrelation.

```
let xf = 1 20 20;
yf = xf'bhat;
yf;
```

Now use the information concerning the autocorrelation.

```
yf = yf + rhohat*(y[20,1] - x[20,.]*bhat);
yf;
```

9.6 Exact Durbin-Watson Statistic

This program contains the code for the program DW.SET which compiles and loads the procedures to compute the exact Durbin-Watson critical value. If you type it into a file, it is usually best to save the file on the subdirectory

`\GAUSS\sp.`

The computations follow those outlined in J. Koerts and A. P. J. Abrahamse, *On the Theory and Application of the General Linear Model*, Rotterdam University Press, 1969, and in J. P. Imhof, "Computing the distribution of quadratic forms in normal variables," *Biometrika*, 1961, Vol. 48, pages 419-426.

```
/*  DW.SET  -- Loads and creates procedures to compute exact
Durbin-Watson critical value */

loadp intquad;
loadp eigsym;
proc INHOFF(u,Lp);
     local e,g,z;
 /*  u is the variable to be integrated
     Lp is the transposed vector of eigenvalues */
     e = (.5*sumc(arctan(Lp'.*u))) ;
     g = exp(sumc(0.25*ln(  1 + (Lp'.*u)^2 ))) ;
     z = sin(e)./(u'.*g);
retp(z');
endp;

proc EXACTDW(dw,x,rho);
  local M,A,Psi,psih,t,w,C,D,A1,val,g;
  t = rows(x);

 /* Creating psi - covariance matrix of disturbances */
  w = seqa(0,1,t);
  psi = reshape(rho^w,t,t);
  psi = shiftr(psi,w,0);
```

```
 psi = psi + (psi - eye(t))';
 psi = psi/(1-rho^2);

/* Computing psi to the 1/2 using the
                              Cholesky decomposition */
 psih = chol(psi);

/* Creating M */
 M = eye(t) - (x*invpd(x'x)*x');

/* Creating A -- matrix with 2's on diagonal,
       -1's on off-diagonal, and 1's at two corners */
 A1 = zeros(t-1,1)~eye(t-1)|zeros(1,t);
 A = eye(t)*2 - (A1 + A1');
 A[1,1] = 1;
 A[t,t] = 1;

/* Computing roots */
 C = psih*M;
 D = C*(A - dw*eye(t))*C';
 L = eigsym(D,0);

/* Integrating */
 val = .5 - (intquad(&IMHOF,0,30,L',40))/pi;

 retp(val);
endp;
```

Chapter 10

Varying Parameter Models

10.1 Introduction

In this chapter you will learn how to use dummy variables to permit some parameters in a regression model to change within the sample period and to test for such a strucural change. You will also be introduced to varying and random coefficient models.

10.2 Use of Dummy Variables in Estimation

The first application of dummy varibles is to account for a change in an intercept value. Equation 10.2.11 describes an investment function which has a larger intercept during the years 1939-1945 which are taken to be the period of World War II.
The values of the exogenous variables x_2 and x_3 are found in the file labelled TABLE10.1 on the disk accompanying this book. LOAD this data, and create the dummy variable which is one during the war years and zero otherwise.

```
load xvar[20,2] = table10.1;
year = seqa(1935,1,20);
d = (year .>= 1939) .and (year .<= 1945);
```

We note in passing that **GAUSS** has three built-in functions that aid in the construction of dummy variables. See your **GAUSS** manual for descriptions of the functions DUMMY, DUMMYBR, DUMMYDN. These will not be used in this Chapter.
Construct the intercept variable, and add it and the dummy varible as columns one and two of the X matrix.

```
x = ones(20,1)~d~xvar;
t = rows(x);
k = cols(x);
```

Construct a (20 x 1) vector of N(0,100) disturbances using the first 20 "official" random numbers[1]

```
sig2 = 100;
open f1 = nrandom.dat;
e = readr(f1,t);
f1 = close(f1);
e = sqrt(sig2)*e;
```

Create the variable y_a in Table 10.1 of the text using the parameter values in Equation 10.2.11.

```
let beta = -10.0 20.0 0.03 0.15;
ya = x*beta + e;
```

Check the data.

```
format 14,6;
ya~x;
```

At the end of chapter 6 we updated our PROC MYOLS. Run that PROC now to place in memory so that we can use it in this chapter. Obtain the OLS parameter estimates for the model described by Equation 10.2.11 and compare to the second column of Table 10.2.

```
{b,covb} = myols(x,ya);
```

Repeat the regression omitting the dummy variable and compare to column 1 of Table 10.2.

```
x1 = ones(20,1)~xvar;
{b1,covb1} = myols(x1,ya);
```

Now create the dummy variable $C_t = 1 - D_t$ and include it in the model with the dummy variable D_t but excluding the overall intercept. Examine the data.

```
c = 1 - d;
x2 = d~c~xvar;
format 14,6;
ya~x2;
```

Obtain the OLS estimates of this model and compare to column 3 of Table 10.2.

```
{b2,covb2} = myols(x2,ya);
```

As noted in the text Equations 10.2.3 and 10.2.8 are equivalent forms. Using the coefficient estimates b_2 we can calculate the coefficient of the dummy variable D_t in (10.2.3) as follows.

[1]These are the ones that came with the original manual. You may construct substitutes for the official ones yourself using the the statement 'e=sqrt(100)*rndn(20,1);'.

```
let r[1,4] = 1 -1 0 0;
delta = r*b2;
delta;
```

Since delta is a linear combination of be, we can compute the standard error of the estimator as

```
sqrt(r*covb2*r');
```

Now consider the possibility that not only is the intercept different during the war years but there is a difference in one or more slope parameters as well. Construct y_b in Table 10.1 using Equation 10.2.15 and examine the data.

```
x = ones(20,1)~d~xvar~(xvar[.,2] .* d);
let beta = -10.0 20.0 0.03 0.15 -0.09;
yb = x*beta + e;
format 10,6;
yb~x;
```

Obtain the OLS results and compare to column 2 of Table 10.4.

```
{b,covb} = myols(x,yb);
```

Now construct the "sets of equations" equivalent model and examine.

```
x1 = d~c~xvar[.,1]~(xvar[.,2] .* d)~(xvar[.,2] .* c);
format 10,6;
x1;
```

Obtain the OLS results and compare to column 3 in Table 10.4.

```
{b1,covb1} = myols(x1,yb);
```

10.3 The Use of Dummy Variables to Test for a Change in the Location Vector

Assuming that XVAR, the dummy variables D_t and C_t and the random disturbances E are still in memory, construct the values y_c in Table 10.1 using Equation (10.3.1) and examine the data.

```
x = ones(20,1)~d~xvar[.,1]~(xvar[.,1] .* d)~
     xvar[.,2]~(xvar[.,2] .* d);
let beta = -10 20 0.03 0.03 0.15 -0.09;
yc = x*beta + e;
format 10,6;
yc~x;
```

Compute the unrestricted and restricted regressions as in Table 10.6 using PROC MYOLS and compute the test statistic u in Equation 10.3.2.

```
{bu,covbu} = myols(x,yc);              /*  unrestricted model   */
sseu = sumc((yc - x*bu)^2);
df = rows(x) - cols(x);
sighat2 = sseu/df;

xr = ones(20,1)~xvar;                  /*  restricted model     */
{br,covbr} = myols(xr,yc);
sser = sumc((yc - xr*br)^2);

u = (sser - sseu)/(3*sighat2);         /*  test statistic       */
pval = cdffc(u,3,df);                  /*  p-value              */

u~pval;
```

10.4 Systematically Varying Parameter Models

This section contains a discussion a general way to incorporate both systematic and/or random parameter variation. To illustrate we will consider an alternative way to model the data generation process of the dependent variable y_b which was introduced in Section 10.2. Instead of including intercept and slope dummy variables we will consider the possiblity that these coefficients vary systematically over time. In particular we will "model" the parameter variation as $\beta_1 = \delta_1 + \delta_2 * \text{year}_t$ and $\beta_2 = \gamma_1 + \gamma_2 * \text{year}_t$. It should be stressed that this specification is incorrect and we are using it only for illustration purposes.
Construct the matrix of explanatory variables following Equations 10.4.2 - 10.4.5.

```
xa = ones(20,1)~year~xvar[.,1]~(xvar[.,1].*year)~xvar[.,2];
```

Estimate this "unrestricted" model and calculate the sum of squared residuals.

```
{ba,covba} = myols(xa,yb);
sseu = (yb - xa*ba)'(yb - xa*ba);
```

Now estimate the "restricted" model, which assumes that $\delta_2 = \gamma_2 = 0$.

```
x = ones(20,1)~xvar;
{b,covb} = myols(x,yb);
```

Compare the restricted and unrestricted estimates. Compute the sum of squared residuals for the restricted model and test the hypothesis that the parameters in question do not vary linearly with time.

```
sser = (yb - x*b)'(yb - x*b);
df = rows(xa) - cols(xa);
u = (sser - sseu)/(2*sseu/df);
pval = cdffc(u,2,df);
u~pval;
```

Thus despite the apparantly substantial changes in the parameter estimates caused by including the variable time, their inclusion does not significantly affect the fit of the model.

10.5 Hildreth-Houck Random Coefficient Models

In this section the essentials of the Hildreth-Houck random coefficient model are presented and an estimation procedure outlined. To illustrate the computational procedure consider the dependent variable y_c considered in Section 10.3. This, of course, is an incorrect assumption for the data generation process and we are using it only for illustration purposes.
Assuming the variable y_c is still in memory, obtain the least squares estimates of the model assuming no parameter variation.

```
x = ones(20,1)~xvar;
{b,covb} = myols(x,yc);
```

Follow the procdure for estimating the parameter covariance matrix Σ discussed below Equation 10.5.7. First calculate the OLS residuals and square them to form `edot`.

```
ehat = yc - x*b;
edot = ehat .* ehat;
```

Construct the idempotent matrix `m`, and square its elements to form `mdot`.

```
m = eye(20) - x*invpd(x'x)*x';
mdot = m .* m;
```

Form the matrices `z` and `f`.

```
z = ones(20,1)~(2*x[.,2])~(2*x[.,3])~(x[.,2]^2)~(2*x[.,2] .* x[.,3])~
     (x[.,3]^2);
f = mdot * z;
```

Estimate the parameters alpha by regressing `edot` on `f`.

```
ahat = edot/f;
```

Form the estimate of the matrix Σ and examine it.

```
sighat = ahat[1]~ahat[2]~ahat[3]|
         ahat[2]~ahat[4]~ahat[5]|
         ahat[3]~ahat[5]~ahat[6];
sighat;
```

As noted in the text there is nothing in the estimation procedure to ensure that this estimated covariance matrix is positive semidefinite. Check the matrix `sighat` by obtaining its determinant.

```
det(sighat);
```

The negative determinant implies that `sighat` is not positive semidefinite, and thus if it were used as a basis for constructing the error variances in Equation 10.5.7 some of those values might be negative. Let us proceed but we will check this possibility. Form the variances in (10.5.7) and examine them.

```
var = z*ahat;
var';
```

Despite the negative variances that appear the estimation procedure is "consistent" and one could simply proceed. For the purposes of completing this example, however, we will set the off-diagonal elements of `sighat` to zero and proceed. We do not suggest this as a general "fix-up" procedure. The reader is advised to consult the cited references if this problem is encountered in practice.

```
sighat = diagrv(eye(3),diag(sighat));
sighat;
```

Now proceed with EGLS estimation.

```
phi = diagrv(eye(20),diag(x*sighat*x'));
covbhat = invpd(x'*invpd(phi)*x);
bhat = covbhat*x'*invpd(phi)*yc;
bhat';
sqrt(diag(covbhat))';
```

As suggested in the text general tests for heteroscedasticity can be used to test for this form of random coefficients. Carry out the Breusch-Pagan test as described in Chapter 9.3.5c.

```
sigtilde = sumc(edot)/20;
lhs = edot/sigtilde;
ahat = lhs/z;
sst = (lhs - meanc(lhs))'(lhs - meanc(lhs));
sse = (lhs - z*ahat)'(lhs - z*ahat);
q = (sst - sse)/2;
q cdfchic(q,cols(z)-1);
```

What do you conclude?

Chapter 11

Sets of Linear Statistical Models

11.1 Introduction

In this chapter you will study two types of models involving sets of linear equations: seemingly unrelated regression equations and pooled time series cross-sectional models.

11.2 Seemingly Unrelated Regression Equations

Begin with the example outlined in Section 11.2.4 in *ITPE2*. It concerns investment for two companies, say General Electric and Westinghouse. LOAD the data from TABLE11.1, check it, and create the the vectors y_1 (column 1 of Table 11.1) and y_2 (column 4 of Table 11.1) containing the investment data. The matrices of independent variables contain a constant term, market values (columns 2 and 5 of Table 11.1), and capital stocks (columns 3 and 6 of Table 11.1.)

```
load dat[20,6] = table11.1;
format 10,7;
dat;
y1 = dat[.,1];
y2 = dat[.,4];
x1 = ones(20,1)~dat[.,2 3];
x2 = ones(20,1)~dat[.,5 6];
```

Estimate each equation separately using least squares. (See Equation 11.2.31 in the text.)

```
b1 = y1/x1;
```

```
b1';
b2 = y2/x2;
b2';
```

Compute the OLS residuals from the regressions, and use them to estimate the contemporaneous covariance matrix. Note that the divisor used is the "average" number of coefficients per equation.

```
e1hat = y1 - x1*b1;
e2hat = y2 - x2*b2;
k1 = rows(b1);
k2 = rows(b2);
t = rows(y1);
df = t - (k1 + k2)/2;
shat = (e1hat~e2hat)'(e1hat~e2hat)/df;
shat;
```

To compute the generalized least squares estimator, `bg`, first combine the data to form one "stacked" model as shown in Equation 11.2.19 in the text.

```
y = y1|y2;
x = (x1~zeros(t,k2) )|( zeros(t,k1)~x2 );
```

Next create the inverse of the estimated error covariance matrix for the stacked error vector. See Equations 11.2.21 and 11.2.28 in the text. The .*. operator produces the Kroneckor product.

```
ishat = invpd(shat);
iphi = ishat .*. eye(t);
```

Now compute the generalized least squares estimate using Equation 11.2.28 in the text.

```
bg = (x'iphi*y)/(x'iphi*x);
bg';
```

This method works fine if the number of observations is small enough. However, the matrix `iphi` has dimensions (MT x MT), where M is the number of equations, and personal computers can quickly run into storage problems. Write a procedure to do the same computations but without creating the entire `iphi` matrix. See Equation ll.2.24 in the text. Note that all computations so far assume that each equation contains the same number of observations.
The procedure SUR takes three arguments. The matrices `x` and `y` contain the data. To use the procedure, the vectors of dependent variables and matrices of independent variables are concatenated horizontally. For example, the `y` matrix has as many columns as there are separate equations and as many rows are there are observations per equation. The third argument, `shat`, contains the estimated contemporaneous covariance matrix. The procedure returns the

variables `xx`, `xty` and `std` so that they can be used for later computations. The name `xty` is given to $x'y$, rather than `xy`, to avoid confusion with the QUICK GRAPHICS XY.
You may want to put the code for the procedure into a file (sur.prc) to rerun at later dates; SUR is used repeatedly in this chapter. Also, be sure that you understand how the **GAUSS** code used represents Equation 11.2.24.

```
proc (4) = sur(x,y,shat);
   local *;

   k = cols(x)/cols(y);
   ishat = invpd(shat);
   xx = (x'x) .* (ishat .*. ones(k,k));
   xty = (x'y) .* (ishat .*. ones(k,1));
   bg =  sumc(xty')/xx;
   std = sqrt(diag(invpd(xx)));
   retp(bg,xx,xty,std);
endp;
```

If the code is in a file, press **F2** to compile and save. Test your procedure by re-estimating the example. The results should be the same as those computed above, `bg`.

```
x = x1~x2;
y = y1~y2;
{bga,xx,xty,std} = sur(x,y,shat);
bga';
bg';
```

As is noted in *ITPE2* another way to estimate the parameters in a seemingly unrelated regressions context is to iterate Equations (11.2.27) and (11.2.28). To compute the iterative generalized least squares estimate, repeatedly estimate the covariance matrix and the coefficients within a DO-LOOP. When the coefficients cease to change (according to the specified tolerance), the estimation is complete. In addition to stopping when convergence has been achieved, it is usually wise to stop after a given number of iterations, since iterative procedures may converge very slowly, if at all.

```
tol = 1;                              /*  initialize tol    */
iter = 1;                             /*  initialize iter   */
format 10,7;                          /*  begin  loop       */
do until ( (tol lt .00001)
         or (iter ge 20) );
  e1hat = y1 - x1*bg[1:k1,1];         /*  new residuals     */
  e2hat = y2 - x2*bg[k1+1:2*k1,1];
  shat = (e1hat~e2hat)'
              (e1hat~e2hat)/t;        /*  new contemp. cov. */
  {bga,xx,xty,std} = sur(x,y,shat);   /*  new estimates     */
```

```
  tol = sumc( abs(bga-bg));          /*  check tol.        */
  iter = iter + 1;                   /*  update iter       */
  bg = bga;                          /*  store current est.*/
  " iter " iter "tol " tol;?;        /*  print             */
endo;                                /*  end loop          */

bga';                                /*  print estimates   */
std';
```

Note that at each iteration the variable `tol` decreases. In the first iteration the starting values were the already obtained SUR estimates. The OLS estimates could have been used and convergence to the same set of estimates obtained. In Chapter 12 more will be said about maximum likelihood estimation. The Iterative estimator of the SUR model is the maximum likelihood estimator of this model if the random disturbances are multivariate normal.
Now carry out a Monte Carlo experiment that illustrates the small sample properties of the SUR estimator. When SUR is based on an estimated contemporaneous matrix the properties of EGLS estimators are asymptotic ones and do not necessarily apply in small samples. Thus in this Monte Carlo experiment we will be interested to see if the SUR estimates are more efficient than the OLS estimates.
In order to simulate the data it must be possible to draw random numbers from a multivariate normal distribution with a given error covariance matrix. Two procedures are given in the Appendix to Chapter 11. We will illustrate the first using characteristic roots and vectors, since that is how the data in Table 11.1 is generated. As an exercise we will replicate the example in the appendix using this method. Note that the **GAUSS** function CHOL performs the Cholesky decompostion.
First obtain the characteristic roots and vectors of the matrix Σ given on page 494.

```
let sigma[2,2] = 1  1
                 1  2;
{d,c} = eigrs2(sigma);
d~c;
```

Note that **GAUSS** orders the characteristic roots and the corresponding characteristic vectors from smallest to largest. To conform to the text order the roots from largest to smallest and reverse the order of the columns of the matrix of characteristic vectors to match.

```
d = rev(d);
c = (rev(c'));
d~c;
```

Form the transformation matrix `sighalf` and compare to the text.

```
sighalf = c * diagrv( eye(2), sqrt(d) );
sighalf;
```

To duplicate the Monte Carlo results in the text we will use the matrices of N(0,1) already stored. LOADM them.

```
loadm e1 = e1nor;
loadm e2 = e2nor;
```

Recall that these matrices are (20 x 250) with each column representing a vector from a multivariate N(0,I) distribution. To create a sample of size T = 20 from a multivariate normal distribution with covariance matrix Σ given in Equation 11.2.33 create a (20 x 2) matrix of N(0,1) values from the first 2 columns of `e1` and apply a transformation like `sighalf` to each row.

```
let sigma[2,2] =  660  175
                  175   90;
{d,c} = eigrs2(sigma);
d = rev(d);
c = (rev(c'))';
sighalf = c * diagrv( eye(2), sqrt(d) );
```

This transformation matrix will serve to transform the N(0,1) random errors to the desired multivariate normal distribution. However, to obtain the numbers in *ITPE2* recall that characteristic vectors are not unique. In fact, if v is a characteristic vector of a matrix then $-v$ is as well. The normalization used in **GAUSS** is the negative of the normalization used to create the data in Table 11.1 of *ITPE2*. While in general the choice does not matter, to replicate the example in the text we change the sign of sighalf, and proceed to transform the random errors.

```
sighalf = -sighalf;
e = e1[.,1 2];
estar = e * sighalf';
```

Define the parameter vectors to be those given in Equation 11.3.2 and construct the vectors `y1` and `y2`. Compare the values to those in Table 11.1.

```
let beta1 = -28.0 0.04 0.14;
let beta2 =  -1.3 0.06 0.06;

y1 = x1*beta1 + estar[.,1];
y2 = x2*beta2 + estar[.,2];
y1~y2;
```

Write a program to simulate data and estimate the least squares and generalized least squares estimates for two equations with equal numbers of coefficients. The two matrices of independent variables, `x1` and `x2`; the two vectors of true parameters, `beta1` and `beta2`; the true contemporaneous covariance matrix, `sigma`; and the number of samples, `nsam`, must be specified. The procedure SUR written above is used within this procedure (MCSUR), so make sure that you have entered it and run it so that it is in memory.
Since this procedure is quite long, you may want to put it into a file: `MCSUR.PRG`.

```
/*   Program to carry out a Monte Carlo experiment for
     the Seemingly Unrelated Regressions model            */

     load dat[20,6] = table11.1;   /*  load data          */
     x1 = ones(20,1)~dat[.,2 3];
     x2 = ones(20,1)~dat[.,5 6];
     T = rows(x1);
     k = cols(x1);
     x = x1~x2;

/*  define parameters    */

     let beta1 = -28.0 0.04 0.14;
     let beta2 =  -1.3 0.06 0.06;
     xb = (x1*beta1)~(x2*beta2);
     nsam=250;

     let sigma[2,2] =  660  175
                       175   90;

 /*  create transformation */

     {d,c} = eigrs2(sigma);
     d = rev(d);
     c = (rev(c'))';
     sighalf= -c * diagrv( eye(2), sqrt(d) );

 /*  load random values   */

     loadm e1 = e1nor;
     loadm e2 = e2nor;

 /*  storage matrix       */

     param = zeros(4*k,nsam);

     i = 1;                  /*  start loop           */
     do until i > nsam;

     if i le 125;            /*  select error vectors */
          c2 = 2*i;
          c1 = c2-1;
          e = e1[.,c1~c2];
     else;
          c2 = 2*(i-125);
          c1 = c2-1;
```

```
     e = e2[.,c1~c2];
endif;

e = e*sighalf';        /*  transform errors     */

y = xb + e;            /*  create y             */

b1 = y[.,1]/x1;        /*  ols                  */
b2 = y[.,2]/x2;
ehat = (y[.,1] - x1*b1)~(y[.,2] - x2*b2);

shat =  ehat'ehat/(t-k) ;       /*   sur        */
{bg,xx,xty,std} = sur(x,y,shat);

param[.,i] = b1|b2|bg;    /*  store estimates   */
i = i + 1;
endo;                     /*  end loop          */

m = meanc(param');     /*  Calculate means      */
std = stdc(param');    /*  Calculate Std Errors */

format 10,5;
(beta1|beta2)~m[1:6,.]~std[1:6,.]~m[7:12,.]~std[7:12,.];

/*    End of Program   */
```

Execute the program and compare the results to Table 11.2 in the text.
Use histograms to compare the distributions of the least squares and GLS estimators of the second parameter in the first equation. To obtain histograms just like those in the top panel of Figure 11.1 we must use the same "breakpoints" as in the text. These are given in the vector `v1` below.

```
let v1 = .0111 .0253 .0395 .0537 .0679;
graphset;
beggraph;
window(1,2);
{b,m,freq} = histp(param[2,.]',v1);
{b,m,freq} = histp(param[8,.]',v1);
endgraph;
```

So that you can reproduce the 2nd, 3rd and 4th panels of Figure 11.1, the breakpoints to use are:

```
let v2 = .0959 .119 .141 .164 .186;
let v3 = .0291 .0446 .0601 .0757 .0912;
let v4 = -.0373 .0121 .0616 .111 .161;
```

In Section 11.2.5 of the text there is a discussion of methods used to test hypotheses in the SUR model. In Section 11.2.6 an example of those tests is given. LOAD the data in the file TABLE11.3 on the disk accompanying this text. Take natural logs to create the dependent and independent variables for each equation. The data consist of thirty observations on prices, quantities and income for three commodities.

```
load dat[30,7] = table11.3;
t = rows(dat);
c = ones(t,1);
lnd = ln(dat);
x1 = c~lnd[.,1 4];
x2 = c~lnd[.,2 4];
x3 = c~lnd[.,3 4];
y1 = lnd[.,5];
y2 = lnd[.,6];
y3 = lnd[.,7];
```

Compute the least squares estimates and compare them to the values in Table 11.4 of *ITPE2*.

```
b1 = y1/x1;
b2 = y2/x2;
b3 = y3/x3;
b1'; b2'; b3';
```

Compute the residuals from the least squares estimates, and estimate the contemporaneous covariance matrix.

```
y = y1~y2~y3;
ehat = y - ((x1*b1)~(x2*b2)~(x3*b3));
k = rows(b1);
shat = ehat'ehat/(t-k);
```

Write a function to compute the squared correlations in Equation 11.2.35.

```
fn corr(i,j) = shat[i,j]^2/(shat[i,i]*shat[j,j]);
```

Using CORR, compute the Lagrange multiplier statistic to test for contemporaneous correlation. See Equation 11.2.34. Under the null hypothesis of no contemporaneous correlation, the test statistic lambda has a χ^2_3 distribution, in this case. Compare to the value given on p.461 of *ITPE2*.

```
lambda = t*(corr(2,1) + corr(3,1) + corr(3,2));
lambda;
cdfchic(lambda,3);
```

Compute the generalized least squares estimates, using the procedure SUR written above in Section 11.2.3. Compare your results to those in Table 11.4.

```
x = x1~x2~x3;
{bg,xx,xty,std} = sur(x,y,shat);
bg~std;
```

Now estimate a restricted model requiring that the price elasticities be the same for all three commodities (the price is the second column of the relevant X matrix). First write the restrictions in matrix format: $R\beta = r$, where β is the vector of unknown parameters.

```
r = zeros(2,9);
r[1 2,2] = 1|1;
r[1,5] = -1;
r[2,8] = -1;
rr = zeros(2,1);
format 2,0;
r~rr;
```

Following Equations 11.2.39 through 11.2.43 in the text, estimate the restricted seemingly unrelated regression estimator. Note that the matrix `xx` was automatically returned from memory when bg was estimated. It is equal to $X'(\hat{\Sigma}^{-1} \otimes I_T)X$ or $\hat{C}^{-1}$ in the text.

```
chat = invpd(xx);
qq = chat*r'invpd(r*chat*r');
bgr = bg + (qq*(rr - r*bg));
```

The covariance matrix of the estimated parameters is:

```
covbgr = chat  - qq*r*chat;
```

Print the results and compare to the third column of Table 11.4.

```
format 8,4;
bgr~sqrt(diag(covbgr));
```

Write a procedure to compute the restricted generalized least squares estimator. It takes as its arguments the output from the unrestricted estimates, `bg` and `xx`, and the matrices describing the constraints, `r` and `rr`.

```
proc (2) = glsr(bg,xx,r,rr);
  local *;
  chat = invpd(xx);
  qq = chat*r'invpd(r*chat*r');
  br = bg + qq*(rr - r*bg);
  covbgr = chat - qq*r*chat;
  retp(br,covbgr);
endp;
```

Use the procedure SUR written above to compute the least squares estimates of the three equations and the matrix `xx`. Do this by setting shat equal to an identity matrix, thereby assuming no contemporaneous correlation. (Note that the computed standard errors will not be correct in this case, but that they are not used in any of the following computations.)

```
shat = eye(3);
{b,xx,xty,std} = sur(x,y,shat);
```

Now compute the restricted generalized least squares (SUR) estimates using `glsr`. Verify that the estimates satisfy the constraints.

```
{br,covbgr} = glsr(b,xx,r,rr);
r*br;
```

Compute the restricted least squares residuals. Use them to compute the contemporaneous covariance matrix.

```
br1 = br[1:3,1];
br2 = br[4:6,1];
br3 = br[7:9,1];
erls = y - ( (x1*br1)~(x2*br2)~(x3*br3) );
shat = erls'erls/(t-k);
```

Compute a new generalized least squares estimate using the new, restricted, estimate `shat`.

```
{bgn,xx,xty,std} = sur(x,y,shat);
```

Now compute the restricted generalized least squares estimate and compare these results to the last column of Table 11.4. Slight differences may be present due to rounding error.

```
{bgrn,covbgrn} = glsr(bgn,xx,r,rr);
bgrn~sqrt(diag(covbgrn));
```

In Section 11.2.7 the problem of unequal numbers of observations in SUR equations is discussed.
Load in the data from Table 11.1 in the text again, this time dropping the last five observations for the first equation.

```
load dat[20,6] = table11.1;
y1 = dat[1:15,1];
x1 = ones(15,1)~dat[1:15,2 3];
y2 = dat[.,4];
x2 = ones(20,1)~dat[.,5 6];
t = rows(y1);
n = rows(y2) - t;
```

Compute the least squares coefficients.

```
b1 = y1/x1;
b2 = y2/x2;
b1';
b2';
```

Compute the generalized least squares estimates, dropping the last five observations for the second group so that the sample sizes are the same. Use the procedure SUR written in Section 11.2.3 above and using the correction in Equation 11.2.60

```
e1 = y1 - x1*b1;
e2 = y2 - x2*b2;
ehat = e1~e2[1:15,1];
shat = ehat'ehat/t;
shat[2,2] = e2'e2/(t+n);
{bb,xx,xty,std} = sur(x1~x2[1:15,.],y1~y2[1:15,.],shat);
shat;
bb';
```

This time adjust `xx` and `xty` to use all observations in the data set. (See Equation 11.2.59 in the text.)

```
x20 = x2[16:20,.];
y20 = y2[16:20,1];
xxadd = (x20'x20)/shat[2,2];
xyadd = (x20'y20)/shat[2,2];
xx[4:6,4:6] = xx[4:6,4:6] + xxadd;
xty = sumc(xty') + ( zeros(3,1)|xyadd );
bg = xty/xx;
bg';
sqrt(diag(invpd(xx)))';
```

11.3 Pooling Time Series and Cross-Sectional Data Using Dummy Variables

In this section you will apply the dummy variable techniques of Chapter 10 to the problem of pooling Time-Series and Cross-Sectional (TSCS) data. LOAD in the data from the file TABLE11.1 on the disk accompanying this book. Columns 1 – 4 contain cost data for four different firms, and columns 5 – 8 contain output data for the four firms. Examine the data.

```
load dat[10,8] = table11.5;
n = 4;
t = rows(dat);
```

```
format 8,4;
dat;
```

Compute the means of each column of data, and use the vector of means, mn, to compute deviations from the mean for each column of data. Call the matrix of deviations from the mean ddat. Note that we can take advantage of **GAUSS**'s elementwise operations to subtract a row from the data matrix. Compare the result to Table 11.6 in *ITPE2*.

```
mn = meanc(dat);
mn;
ddat = dat - mn';
ddat;
```

Take the observations on cost (deviations from the mean) and put them in a single column vector using the **GAUSS** function VEC. Do the same for the observations on output. This stacks the data on the slope variables as in Equation 11.4.13–11.4.16

```
dy = ddat[.,1:4];
w = vec(dy);
dx = ddat[.,5:8];
z  = vec(dx);
```

Compute the slope coefficient using least squares. Remember that this estimate of the slope coefficient is equivalent to a least squares regression with dummy variables for each firm (omitting the constant term).

```
bs = w/z;
bs;
```

Compute the N intercept terms using Equation 11.4.10 and the means computed above.

```
b1 = mn[1:4,1] - mn[5:8,1]*bs;
b1';
```

Compute the estimate of the error variance which is assumed to be constant across the firms. Use Equation 11.4.18.

```
k1 = rows(bs);
ehat = w - z*bs;
sse = ehat'ehat;
sighat2 = sse/(n*t - n - k1);
sighat2;
```

Estimate the variance of the estimated slope parameter and its standard deviation.

```
covbs = sighat2*invpd(z'z);
sqrt(covbs);
```

Your results should agree with those on pages 477-478 of the text. As an exercise, apply OLS (use PROC MYOLS) to the model including the dummy variables for the intercepts to verify, in this small model, that the results are the same. First create y vector and X matrix as shown in Equations 11.4.4 and 11.4.5.

```
y = dat[.,1:4];
y = vec(y);
xs = dat[.,5:8];
xs = vec(xs);
x = ( eye(n) .*. ones(t,1) )~xs;
```

Now make sure PROC MYOLS is in memory or can be found by the **GAUSS** auto-load feature.

```
{b,covb} = myols(x,y);
```

Next, carry out the F-test for the hypothesis that all the intercepts are equal. Compute the least squares coefficients this time constraining all of the intercepts to be equal. Use the vector y just created and add a column of ones to `xs` to create the X matrix. model.

```
x = ones(n*t,1)~xs;
b = y/x;
b';
```

Compute the sum of squared residuals from the restricted regression and call it `sser`.

```
ehatr = y - x*b;
sser = ehatr'ehatr;
sser;
```

Test the null hypothesis that the intercepts are equal.

```
df1 = n-1;
df2 = n*t - n - k1;
fstat = (sser - sse)/(df1*sighat2);
fstat;
cdffc(fstat,df1,df2);
```

11.4 Pooling Time Series and Cross-Sectional Data Using Error Components

In this section the assumption is made that the intercepts of each firm are random instead of fixed, and consist of a "mean" intercept and a random component. The appropriate estimator for this model is the genreralized least squares estimator, and the steps involved are summarized on pages 485-486 of *ITPE2*.

1. Calculate the dummy variable estimator, which is simply the estimator of the slope parameter (`bs`) computed in the previous section.

```
bs';
```

2. Use the residuals from (1) to calculate the unbiased estimate of the error variance. Again, this we have calculated above.

```
sighat2;
```

3. Estimate beta using the observations on the individual means, as in (11.5.27).

```
mny = mn[1:4,1];
mnx = ones(4,1)~mn[5:8,1];
bstar = mny/mnx;
bstar';
```

4. Compute the residuals from (3). Use these residuals to calculate (11.5.28)

```
k = rows(bstar);
vstar = mny - mnx*bstar;
sig1 = t * (vstar'vstar)/(n - k);
sig1/t;
```

5. Use the estimates `sig1` and `sighat2` to estimate σ_u^2 as in Equation 11.5.30.

```
sigu2 = (sig1 - sighat2)/t;
sigu2;
```

   If `sigu2` is negative, then the researcher is advised to rethink the model formulation rather than proceeding in an ad hoc manner.

6. Calculate $\hat{\alpha}$

```
alphahat = 1 - sqrt(sighat2/sig1);
alphahat;
```

7. Transform the data by subtracting the fraction α of their means.

```
sdat = dat - alphahat*mn';
sy = vec(sdat[.,1:4]);
sx = vec(sdat[.,5:8]);
```

8. Add an intercept to the matrix of explanatory variables and obtain the EGLS estimator of the variance components model by applying OLS to the transformed data. Note that the intercept must be transformed as well.

```
sx = (ones(n*t,1)-alphahat)~sx;
bhathat = sy/sx;
bhathat';
sehat = sy - sx*bhathat;
sighate = sehat'sehat/(rows(sx) - cols(sx));
covbhat = sighate * invpd(sx'sx);
stderr = sqrt(diag(covbhat));
stderr';
```

Next we consider the problem of "predicting" the random components. Compute the residuals using the generalized least squares estimator. Then reshape the column vector of residuals into a (N x T) matrix so that the first column contains the residuals for the first firm, the second column for the second firm, etc. To predict the random components, sum the residuals for each firm and multiply by σ_u^2/σ_1^2 as in Equation 11.5.31.

```
ehathat = y - x*bhathat;
ui = (sigu2/sig1)*sumc(reshape(ehathat,n,t)');
ui';
```

Compare these answers with the corrsponding estimates from the dummy variable estimator by taking the deviation from the means of the dummy variables as suggested near the bottom of page 488. The estimated intercepts `b1` should stillbe in memory.

```
uidv = b1 - meanc(b1);
uidv';
```

Finally, test the specification by testing whether individual components exist. Calculate the value of the Lagrange multiplier test statistic to test the null hypothesis that σ_u^2 is equal to zero. See Equation 11.5.32 in the text.

```
eje = sumc( sumc(reshape(ehatr,n,t)')^2 );
ratio = eje/(ehatr'ehatr);
lambda = (n*t/(2*(t-1))) * (ratio - 1)^2;
lambda;
cdfchic(lambda,1);
```

11.5 The Choice of Model for Pooling

In this section a variety of factors are presented for consideration when trying to decide which model, the dummy variable or error components model, to use in a particular application. One factor to consider is whether the random error components are correlated with the X data using the Hausman specification error test.
Compute the Chi-square statistic for the Hausman test given in Equation 11.6.2 in the text. Rejection of the null hypothesis that the slope coefficients are the same in the two models suggests that the error components model is not appropriate.

```
k = rows(bhathat);
bdif = (bs - bhathat[2:k,1]);
mdif = covbs - covbhat[2:k,2:k];
m = bdif'invpd(mdif)*bdif;
m;
cdfchic(m,k-1);
```

What do you conclude?

Chapter 12

Estimating Nonlinear Models

12.1 Introduction

This chapter deals with models that are intrinsically nonlinear. That is, they are nonlinear in the parameters and there is no way to transform them to being linear in the parameters. In such cases nonlinear least squares (NLS) and maximum likelihood (ML) estimation techniques are appropriate, depending on error assumptions. In both cases the first order conditions of the optimization problem are nonlinear and thus numerical techniques are relied upon to minimize the sum of squared errors or maximize the likelihood function. Thus in this Chapter the numerical optimization techniques are presented and asymptotic properties of the estimators stated.

12.2 Principles of Nonlinear Least Squares

Consider the problem of estimating the parameter in Equation (12.2.1) by nonlinear least squares. LOAD the data from file TABLE12.1 on the disk with this book and check it.

```
load dat[20,3] = table12.1;
y = dat[.,1];
x = dat[.,2 3];
format 8,4;
y~x;
```

Write a PROC to calculate the values of the function (12.2.1), given that `x` is in memory, for any value of the unknown parameter. Write a FUNCTION to calculate the residual sum of squares `rsq` (12.2.2) again assuming that `x` is in memory.

```
proc FNB(beta);
     retp( x[.,1] .* beta + x[.,2] .* (beta^2));
endp;

fn rsq(beta) = sumc( (y - fnb(beta)).*(y - fnb(beta)) );
```

The reason for treating the functions differently is that the **GAUSS** functions GRADP and HESSP, which calculate numerical first and second derivatives of a function, take as arguments PROCs but not FUNCTIONs. We will demonstrate the use of these functions in this chapter.
Plot the values of the sum of squares function, as in Figure 12.1 of the text, for values of β starting at -3.

```
beta = seqa(-3,.1,56);

library qgraph;
xy(beta,rsq(beta'));
```

In order to use the Gauss-Newton algorithm to estimate the parameters we need the derivative of the function with respect to the parameter as in (12.2.26). Write a PROC to calculate that derivative assuming `x` is in memory.

```
proc DERV(beta);
     retp( x[.,1] + 2*beta*x[.,2] );
endp;
```

Use the Gauss-Newton algorithm to estimate the parameter, using the initial value of 4. Note that the stopping rule is based on the number of iterations or convergence of the algorithm to a specified tolerance. The steps of the Gauss-Newton algorithm are defined in Equation (12.2.29).

```
b1 = 4;
crit = 1;
iter = 1;
format 10,4;
do until (iter gt 25) or (crit < 1e-8);
     iter b1 rsq(b1);

     /*  Equation 12.2.29   */

     bn = b1 +
        (DERV(b1)'(y - FNB(b1)) )/( DERV(b1)'DERV(b1) );

     crit = abs(bn - b1);
     b1 = bn;
     iter = iter + 1;
endo;
```

Try different starting values and note the different rates of convergence and that the process converges to different points depending on the starting value used. This should give ample warning that in nonlinear problems one must try a variety of starting values to ensure that the global minimum of `rsq` is found.
Below is PROC `NLSG` which does nonlinear least squares estimation of a non-linear regression model. It takes as arguments the starting values b0; the right-hand- side of the regression model, $E(y)$, which is passed as a PROC, `&FI`; the analytic gradient vector which is also passed as a PROC, `&GRAD`, and the data matrices `x` and `y`.
In addition it has a step-length adjustment which halves the step length until no further reduction in the sum of squared errors is possible.
You should place this PROC in a file for future use and run it to place it in memory.

```
*************************************************************/
* PROC NLSG ---  Non-linear Least Squares                    */
*                using analytic derivatives                  */
*************************************************************/

proc (2) =  NLSG(b0,&FI,&GRAD,x,y);

    local FI:proc, GRAD:proc;
    local *;


    crit = 1;                       /*  Initialize values      */
    iter = 1;
    k = rows(b0);
    s = 0;

 do until (crit < 1e-8) or (iter > 25);   /*  Begin loop        */


    z = GRAD(b0);                  /*  Evaluate derivatives    */
    u = y - FI(b0);                /*  Compute residuals       */
    sse = u'u;                     /*  Compute sum of sqrd res.*/
    crit = solpd(z'u,z'z);         /*  Compute full step adjust*/
    gosub step;                    /*  Compute step length     */
    b = b0 + s*crit;               /*  Update parameters       */
    gosub prnt;                    /*  Print iteration results */
    iter = iter + 1;               /*  Update for iteration    */
    crit = abs(b - b0);            /*  check convergence       */
    b0 = b;                        /*  store new value of est. */

 endo;                             /*  end loop                */
```

```
    /*  Compute covariance matrix */

    sighat2 = sse/(rows(y) - k);
    covb = sighat2*invpd(z'z);

    /*  Print out final results   */

    ?;
    "Final Results: ";
    ?;
    "Coefficients :  "  b0';
    "Std. Errors  :  "  sqrt(diag(covb))';
    "Sighat2      :  "  sighat2;

retp(b0,z);

step:                           /*  Subroutine for step length */
   s = 2;
   ss1 = 2; ss2 = 1;
   do until ss1 <= ss2;
       s = s/2;
       u1 = y - FI(b0 + s*crit);
       u2 = y - FI(b0 + s*crit/2);
       ss1 = u1'u1;
       ss2 = u2'u2;
   endo;
return;

prnt:                           /*  Subroutine for printing   */
   format 4,2;   " i = " iter;;
   format 4,2;   " Steplength = " s;;
   format 10,6;  " SSE = " sse;
                 " b = " b0';?;
return;

endp;
```

Use PROC NLSG to estimate β using the 4 starting values in Table 12.2.

```
{bn,z} = NLSG(4,&FNB,&DERV,x,y);
{bn,z} = NLSG(-3,&FNB,&DERV,x,y);
{bn,z} = NLSG(-1.05,&FNB,&DERV,x,y);
{bn,z} = NLSG(-.9,&FNB,&DERV,x,y);
```

The procedure returns not only the estimated parameter but also z. Thus we can calculate the estimate of the error variance.

```
sighat2 = rsq(bn)/(rows(y) - rows(bn));
sighat2;
```

The asymptotic variance of the estimated parameter.

```
varb = sighat2*invpd(z'z);
varb;
```

And construct a 95% confidence interval for the regression parameter β.

```
interval = 1.96*sqrt(varb);
(bn - interval) (bn + interval);
```

This problem was relatively simple as the analytic gradient was easy to derive and program. It will not always be so easy. The following PROC again obtains NLS parameter estimates, but it calculates numerical gradients. In this version the computation of the numerical derivative is shown explicitly, but the **GAUSS** function GRADP could be used instead. Put PROC `NLS` in a file and run it to put in memory.

```
/*************************************************************/
/*   PROC NLS  ---  Non-linear Least Squares                 */
/*                  using numerical derivatives              */
/*************************************************************/

proc (2) =  NLS(b0,&FI,x,y);

     local FI:proc;
     local *;
     crit = 1;                      /*  Initialize values      */
     iter = 1;
    s = 0;
    k = rows(b0);

    dh = 1e-6;                     /*  Values used in gradients*/
    e = eye(k)*dh;

do until (crit < 1e-8) or (iter > 25);   /*  Begin do loop  */

    /*  Compute numerical gradients  */
    /*  Could use z = gradp(&FI,b0); */

    z = (FI(b0 + e) - FI(b0 - e))/(2*dh);


    u = y - FI(b0);              /*  Compute residuals         */
    sse = u'u;                   /*  Compute sum of sqrd res.  */
```

```
    crit = solpd(z'u,z'z);       /*  Compute full step adjust  */
    gosub step;                  /*  Compute step length       */
    b = b0 + s*crit;             /*  Update parameters         */
    gosub prnt;                  /*  Print results for iter'n  */
    iter = iter + 1;             /*  Update for next iter'n    */
    crit = abs(b - b0);
    b0 = b;

 endo;

    /*  Compute covariance matrix   */

    sighat2 = sse/(rows(y) - k);
    covb = sighat2*invpd(z'z);

    /*  Print out final results     */

    " Final Results:  ";
    " Coefficients :  "  b0';
    " Std. Errors  :  "  sqrt(diag(covb))';
    " Sighat2      :  "  sighat2;

retp(b0,z);

step:                          /*  Subroutine for step length  */
   s = 2;
   ss1 = 2; ss2 = 1;
   do until ss1 <= ss2;
       s = s/2;
       u1 = y - FI(b0 + s*crit);
       u2 = y - FI(b0 + s*crit/2);
       ss1 = u1'u1;
       ss2 = u2'u2;
   endo;
return;

prnt:                           /*  Subroutine for printing    */
   format 4,2;   " i = " iter;;
   format 4,2;   " Steplength = " s;;
   format 10,6;  " SSE = " sse;
                 " b = " b0';?;
return;

endp;
```

Now, repeat the nonlinear estimation of our model using PROC NLS for the same set of starting values.

```
{bn,z} = NLS(4,&FNB,x,y);
{bn,z} = NLS(-3,&FNB,x,y);
{bn,z} = NLS(-1.05,&FNB,x,y);
{bn,z} = NLS(-0.9,&FNB,x,y);
```

In sections 12.2.2 and 12.2.3 of ITPE2 the general multiple parameter nonlinear regression model is presented and examples of Cobb-Douglas and CES production functions given. LOAD the data for both examples from file TABLE12.3 on the data disk and check it.

```
load dat[30,3] = table12.3;
y = dat[.,3];
x = dat[.,1 2];
y~x;
```

Write PROC CD to calculate $E(y)$ for the Cobb-Douglas production function shown in Equation (12.2.45) for the given x data. PROC CD returns a (T x 1) vector.

```
proc CD(b);
     retp((b[1,.] .* (x[.,1]^b[2,.]) .* (x[.,2]^b[3,.])));
endp;
```

Write a PROC to calculate the gradient vector, Equation (12.2.47), for each of the data points. Note that this returns a (T x K) matrix.

```
proc GRADCD(b);
     local g;
             g = ( CD(b)  ./ b[1,.] )~
                 ( ln(x[.,1]) .* CD(b) )~
                 ( ln(x[.,2]) .* CD(b) );
     retp(g);
endp;
```

Now estimate the parameters of the model using the starting values (1,0,0) and using analytic derivatives. The results will be identical to those in the text except for the estimate of the error variance. In Equation (12.2.43b) the text uses (T - K) as the divisor, as we have, but the numerical results in the book use T as the divisor.

```
let b0 = 1 0 0;
{b,z} = NLSG(b0,&CD,&GRADCD,x,y);
```

Repeat the exercise using numerical derivatives.

```
{b,z} = NLS(b0,&CD,x,y);
```

The second example in the text is the CES production function. Write a PROC for $E(y)$, Equation (12.2.53), and the gradient (12.2.54). Note that both PROCs assume that `x` is in memory.

```
proc CES(b);
     local m1,m2,mid;
     m1 = x[.,1]^b[3,.];
     m2 = x[.,2]^b[3,.];
     mid = (b[2,.].*m1) + ( (1-b[2,.]).*m2 );
     retp( b[1,.] + (b[4,.] .* ln(mid)) );
endp;

proc GRADCES(b);
     local g2,g3,mid;
     mid = (b[2,.] .* (x[.,1]^b[3,.])) +
           ((1 - b[2,.]) .*  x[.,2]^b[3,.]);

     g2 = b[4,.] .* (x[.,1]^b[3,.] - x[.,2]^b[3,.]) ./ mid;

     g3 = b[4,.] .* (ln(x[.,1]).*b[2,.].*x[.,1]^b[3,.]
          + ln(x[.,2]).* (1-b[2,.]).*x[.,2]^b[3,.]) ./ mid;

     retp( ones(rows(y),1)~g2~g3~ln(mid) );
endp;
```

As you can see, the process of deriving and programming the gradient vector can become a sizable problem. Estimate the parameters of the CES production function. The dependent variable is the natural log of y and take as starting values (1, .5, -1, -1).

```
y = ln(y);
let b0 = 1 .5 -1 -1;
{b,z} = NLSG(b0,&CES,&GRADCES,x,y);
```

Repeat the estimation using numerical derivatives.

```
{b,z} = NLS(b0,&CES,x,y);
```

In Section 12.2.4 the Newton-Raphson algorithm is introduced. It is a general purpose optimization algorithm that uses first and second derivatives. The algorithm is illustrated with the one parameter model in Equation (12.2.1). LOAD the data from file TABLE12.1.

```
load dat[20,3] = table12.1;
y = dat[.,1];
x = dat[.,2 3];
x1 = dat[.,2];
x2 = dat[.,3];
```

Write functions to evaluate the function value, Equation 12.2.1, its first and second derivatives, and the sum of squared errors.

```
fn FNB(beta) = x1 .* beta + x2 .* (beta^2);
fn DERV1(beta) = x1 + 2*beta*x2;
fn DERV2(beta) = 2*x2;
fn RSQ(beta) = sumc( (y - FNB(beta)).*(y - FNB(beta)) );
```

Write a PROC that computes the parameter estimates using the Gauss-Newton first and then using the Newton-Raphson procedure. Put this PROC in a file and run it.

```
proc GNNR(bn);
local b1,num,gn,nr,flag,bn0,crit,iter;
bn0 = bn;                          /*  bn0 is starting value  */
flag = 1;
do until flag > 2;
     bn = bn0;
     crit = 1;
     iter = 1;

do until (crit < 1e-6) or (iter > 25);
     b1 = bn;
     iter;; b1;; rsq(b1);

     num = DERV1(b1)'(y-FNB(b1));  /*  numerator of (12.2.74) and
                                       (12.2.29)                 */

     gn = DERV1(b1)'DERV1(b1);         /*  denom. of (12.2.29)  */
     nr = gn - (y-FNB(b1))'DERV2(b1);  /*  denom. of (12.2.74)  */

     if flag == 1;
          bn = b1 + (num/gn);        /* full Gauss-Newton step  */
     elseif flag == 2;
          bn = b1 + (num/nr);       /* full Newton-Raphson step */
     endif;

     iter = iter + 1;
     crit = abs(bn - b1);
endo;
     flag = flag + 1;?;
endo;
retp("");
endp;
```

Now use PROC GNNR to replicate Table 12.4.

```
GNNR(1);
GNNR(-2);
GNNR(0.1);
GNNR(-.9);
```

12.3 Estimation of Linear Models with General Covariance Matrix

In this section the general linear model is considered. Maximum likelihood and nonlinear least squares estimators, as well as Bayesian techniques, are discussed and applied to models of autocorrelation and heteroskedasticity.
The first example is given in Section 12.3.2 where a variety of estimators are developed for the first-order autoregressive error model. LOAD the data in file TABLE9.2 and examine it. This data was used first in Section 9.5.6.

```
load dat[20,3] = table9.2;
t = rows(dat);
y = dat[.,1];
x = ones(t,1)~dat[.,2 3];
k = cols(x);

format 8,4;
y~x;
```

Obtain the least squares estimates.

```
b = y/x;
b';
```

Write a PROC to obtain the estimate of rho in Equation (9.5.40). Note that PROC `NEWRHO`, like many others in this section, assume that `y` and `x` are in memory. If you try to run the proc before `y` and `x` are defined you will get an error message, as **GAUSS** will assume they are uninitialized procs.

```
proc NEWRHO(b);
     local e;
     e = y - x*b;
     retp( e[2:t,.]/e[1:t-1,.] );
endp;
```

Run the PROC to load it into memory and use it to estimate ρ.

```
rhohat = newrho(b);
rhohat;
```

Write a PROC to obtain the EGLS estimates of β, the sum of squared errors and the transformed X matrix.

```
proc (3) = NEWB(rho);
local dstar,ystar,sse,bstar,xstar,data;

     /*   Transform the data    */

             data = y~x;
             dstar = data - (rho*(zeros(1,k+1)|data[1:t-1,.]));
             dstar[1,.] = sqrt( 1 - rho^2)*data[1,.];
             ystar = dstar[.,1];
             xstar = dstar[.,2:k+1];

     /*  Obtain the estimates   */

             bstar = ystar/xstar;
             sse = (ystar - xstar*bstar)'(ystar - xstar*bstar);

             retp( bstar,sse,xstar );
endp;
```

Use the PROC to obtain the EGLS parameter estimates and then the estimated covariance matrix.

```
{bstar,sse,xstar} = newb(rhohat);
sighat2 = sse/(t - k);
covb = sighat2*invpd(xstar'xstar);
```

In Equations (12.3.35) - (12.3.37) are the approximate asymptotic covariance matrices for the estimates of β, σ^2 and ρ. Use these to calculate "asymptotic standard errors" and print out the results. Compare these results to the last row of Table 12.5.

```
"bhat rhohat sighat2  "  bstar';;rhohat;;sighat2;
"std.err.             "  sqrt(diag(covb))';;sqrt((1-rhohat^2)/t);;
                         sqrt(2*(sighat2^2)/t);
```

An alternative to EGLS is to apply NLS to Equation (12.3.28). This approach discards the first observation which dramatically alters the estimates, despite the fact that asymptotically it does not matter. Methods that retain the first observation are more efficient in small samples and are preferred. But for comparison purposes this is a useful exercise. First transform the data.

```
y  = dat[2:t,1];
yl = dat[1:t-1,1];
x  = ones(t-1,1)~dat[2:t,2:k];
xl = ones(t-1,1)~dat[1:t-1,2:k];
y~x;
```

Write a PROC to calculate $E(y)$ given initial values of β and ρ stacked into the vector b0.

```
proc AUTO(b0);
     local m1,m2,m3,m4,m5;

     m1 = yl .* b0[4,.];
     m2 = (x[.,1] .* b0[1,.]) + (x[.,2] .* b0[2,.])
          + (x[.,3] .* b0[3,.]);
     m3 = (xl[.,1] .* b0[1,.]) .* b0[4,.];
     m4 = (xl[.,2] .* b0[2,.]) .* b0[4,.];
     m5 = (xl[.,3] .* b0[3,.]) .* b0[4,.];
     retp(m1+m2-m3-m4-m5);

endp;
```

Make sure that NLS is in memory, and use it to obtain estimates.

```
b0 = b|rhohat;
{b,z} = NLS(b0,&AUTO,x,y);
```

Compare these NLS estimates to those in Table 12.5 and you will see that they are substantially different.
In Equations (12.3.29) and (12.3.30) outline the Cochrane-Orcutt procedure. Write a PROC that uses this algorithm but retaining the first observation.

```
proc (3) = CORC(rho);
     local iter,crit,b,rho1,sse,sighat2,covb;
iter = 1;
crit = 1;
do until (crit < 1e-6) or (iter > 25);

     "iteration " iter " rho "  rho " crit "  crit;
     {b,sse,xstar} = newb(rho);
     rho1 = newrho(b);
     crit = abs(rho1 - rho);
     rho = rho1;
     iter = iter + 1;
endo;
      sighat2 = sse/(t - k);
      covb = sighat2 * invpd(xstar'xstar);
      "SSE"  sse;
      "Est    "  b';;rho;;sighat2;
      "Std err"  sqrt(diag(covb))';;sqrt((1-rho^2)/t);;
                                    sqrt(2*(sighat2^2)/t);
     retp(b,sse,covb);
endp;
```

Reconstruct the original y and X.

```
y = dat[.,1];
x = ones(t,1)~dat[.,2:k];
t = rows(x);
k = cols(x);
```

Use PROC CORC to obtain NLS estimates starting from several initial values of ρ.

```
{b,sse,covb} = CORC(0);
{b,sse,covb} = CORC(.5);
{b,sse,covb} = CORC(.9);
```

Compare these estimates to those in the first row of Table 12.5.
Maximum likelihood estimation of the parameters of this model can proceed in several ways. In Equation (12.3.33) gives, except for constants, the expression for the concentrated likelihood function, which is only a function of ρ. One possiblity is to search over values of ρ and choose the value that maximizes (12.3.33). To that end write a PROC that calculates this likelihood given a value of ρ and with y and X in memory. Note that this proc returns twice the "elements" of the log-likelihood given in (12.3.10) and not the sum.

```
/*************************************************************/
/*   PROC AUTOLIC --  Concentrated Log-Likelihood            */
/*                    for AR(1) errors                       */
/*************************************************************/

proc AUTOLIC(rho);
     local l1,k,T,dstar,ystar,dat,estar,phi,sigmasq,b,xstar;

     k = cols(x);
     t = rows(x);

     /* Transform the data  */

     dat = y~x;
     dstar = dat - (rho*(zeros(1,k+1)|dat[1:T-1,.]));
     dstar[1,.] = sqrt( 1 - rho^2)*dat[1,.];
     ystar = dstar[.,1];
     xstar = dstar[.,2:k+1];

     /* Compute b and sigmasq             */
     /* See Equations 12.3.11 and 12.3.12 */

     b = ystar/xstar;
     estar = ystar - xstar*b;
```

```
    sigmasq = (estar'estar)/T;

    /* Compute components of the likelihood function */

    phi = 1/(1 - rho^2);
    l1 = ln(2*pi*sigmasq) + ln(phi)/T;
    retp( -l1 - ((estar.*estar)/sigmasq) );

endp;
```

Note that this PROC returns a (T x 1) vector of components of the log-likelihood. Search over the [0,1) interval first in units of .01.

```
rhov = seqa(0,.01,100);
l = zeros(100,1);
iter = 1;
do while iter le 100;
     rho = rhov[iter,1];
     l[iter,1]=sumc(AUTOLIC(rho));
     iter = iter + 1;
endo;
```

Find the value of rho corresponding to the maximum and then search in a finer grid over the neighborhood about this value.

```
rhoml = rhov[maxindc(l),1];
rhoml = rhoml -.1;
rhov = seqa(rhoml,.001,200);
l = zeros(200,1);
iter = 1;
do while iter le 200;
     rho = rhov[iter,1];
     l[iter,1] = sumc(AUTOLIC(rho));
     iter = iter + 1;
endo;
rhoml = rhov[maxindc(l),1];

format 8,4;
rhoml;
```

Once the maximum likelihood estimate of rho is obtained the ML estimates of beta can be found using PROC `NEWB`.

```
{b,sse,xstar} = NEWB(rhoml);
sighat2 = sse/t;
```

Since we will find the ML estimates in several ways, write a PROC to construct and print out the final results with their asymptotic standard errors, given that the estimates of β, ρ and σ^2 have been stacked into a vector, `param`.

```
proc AUTOCOV(param);
     local k,t,rho,b,sigmasq,dstar,ystar,xstar,varrho,varsig,data;

     data = y~X;
     k = rows(param)-2;
     t = rows(data);
     b = param[1:k,1];
     rho = param[k+1,1];
     sigmasq = param[k+2,1];
     dstar = data - (rho*(zeros(1,k+1)|data[1:T-1,.]));
     dstar[1,.] = sqrt( 1 - rho^2)*data[1,.];
     ystar = dstar[.,1];
     xstar = dstar[.,2:k+1];
     covb = sigmasq*invpd(xstar'xstar);
     varrho = (1-rho^2)/t;
     varsig = 2*(sigmasq^2)/t;
     "Final results";?;
     "Est          " b';; rho;; sigmasq;
     "Std. errs.   " sqrt(diag(covb))';;sqrt(varrho);;sqrt(varsig);

     retp(covb);
endp;
```

Use this PROC to print out final results for the ML estimates just obtained.

```
param = b|rhoml|sighat2;
covb = AUTOCOV(param);
```

The search procedure is effective but potentially costly and does not ensure that the global maximum of the likelihood function is obtained. Another alternative is to maximize the Log-likelihood function using a general optimization procedure. Several of these are described in Section 12.2.5 of the text. While it is possible to obtain analytical expressions for the first and second derivatives it is tedious to do so and then program them. Below is a ML procedure that uses the method of Berndt-Hall-Hall-Hausman (BHHH) with numerical first derivatives. It assumes y and X are in memory and initial estimates and the PROC defining the "components " of the likelihood function are passed to it (NOTE: not the summed log-likelihood).

```
/*************************************************************/
/*    PROC MAXL -- Maximum likelihood estimation (BHHH)      */
/*************************************************************/

proc MAXL(b0,&LI,x,y);
     local LI:proc;
     local *;
```

```
     db = 1;                  /* Initialize values           */
     iter = 1;
     s = 0;

 do until db < 1e-5;          /* Begin do loop               */
     z = gradp(&LI,b0);       /* Compute gradients (T x K)   */
     H = z'z;                 /* Compute approx. to Hessian  */
     g = -sumc(z);            /* Gradient vector (K x 1)     */
     db = -inv(H)*g;          /* Compute full step adjustment */
     gosub step;              /* Compute step length         */
     b = b0 + s*db;           /* Update parameters           */
     gosub prnt;              /* Print results for iteration */
     iter = iter + 1;         /* Update for next iteration   */
     db = abs(b-b0);          /* Convergence criteria        */
     b0 = b;                  /* Replace old estimate        */
  endo;

  std = sqrt(diag(invpd(H))); /* Compute estimated std. errors*/
  ?;                          /* Print out final results     */
  "Final Results:  ";
  "  Parameters:   " b0';
  "  Std. Errors:  " std';

  retp(b0);                   /* Return parameters to memory */

  step:                       /* Subroutine for step length  */
    s = 2;
    li1 = 0; li2 = 1;
    do until li1 >= li2;
       s = s/2;
       li1 = sumc(LI(b0 + s*db));
       li2 = sumc(LI(b0 + s*db/2));
    endo;
  return;

  prnt:                       /* Subroutine for printing     */
    format 4,2; "i = " iter;;
    format 4,2; "  Steplength = " s;;
    format 10,6; " Ln Likelihood = " sumc(LI(b0));
    format 10,6; " Parameters: " b0';
  return;
endp;
```

To use this general purpose algorithm we must write a PROC that defines the components of the log-likelihood function of the first-order autoregressive model. Its argument is the stacked vector of parameter values for β, ρ and then

σ^2. PROC AUTOLI is fairly long, so place it in a convenient file and run it.

```
/*****************************************************************/
/*                                                               */
/*    PROC AUTOLI -- Log-likelihood with AR(1) errors            */
/*****************************************************************/

proc AUTOLI(param0);
     local l1,k,T,b,rho,e,elag,estar,zz,phi,sigmasq;

/* Take apart the parameter vector */

     k = rows(param0);
     t = rows(y);
     b = param0[1:k-2,1];
     rho = param0[k-1,1];
     sigmasq = param0[k,1];

/* Compute the transformed error term */

     e = y - x*b;
     elag = 0|e[1:t-1,1];
     estar = e - (rho*elag);
     estar[1,1] = sqrt(1 - (rho^2))*e[1,1];

/* Compute the components of the likelihood function */

     phi = 1/(1 - (rho^2));
     l1 = ln(2*pi*sigmasq) + ln(phi)/t;
     retp( - l1 - ((estar.*estar)/sigmasq)  );
endp;
```

Set the initial parameter values and apply the proc. In general several sets of starting values should be tried since the procedure may converge to a local maximum or even a minimum of the log-likelihood function.

```
let b0 = 4  2 .7 .5 7;
b = MAXL(b0,&AUTOLI,x,y);
covb = AUTOCOV(b);
```

In Section 12.3.2c the principles of Bayesian methodology are applied to the autocorrelation problem. The example used is based on the model and data from Section 9.5.6. LOAD the data from that section and inspect the data.

```
load dat[20,3] = table9.2;
t = rows(dat);
k = cols(dat);
```

```
nu = t - k;
y = dat[.,1];
x = ones(t,1)~dat[.,2 3];
format 8,4;
y~x;
```

The "kernel" of the posterior p.d.f. for the autocorrelation parameter, ρ, is given in Equation 12.3.41. The normalizing constant is given in the form of an integral in Equation 12.3.42. PROC `RHOKERN` calculates the value of kernel for a vector, or matrix, of ρ values. It uses PROC `NEWB` to calculate the EGLS estimator of β (`bstar`), the sum of squared errors (`sse`) and the transformed X matrix (`xstar`). The PROC is written in a way such that it can be used by **GAUSS**'s numerical integration function INTQUAD1. That is, the proc must return a vector or matrix the same size as the single argument of the proc. See your **GAUSS** manual for an example. Before you place `RHOKERN` in memory make sure PROC `NEWB` is in memory or can be automatically loaded by **GAUSS**.

```
proc RHOKERN(rho);
     local m,n,store,i,j,bstar,sse,xstar;

     m = rows(rho);          /*  define dimensions of rho  */
     n = cols(rho);
     store = zeros(m,n);     /*  initialize storage matrix */
     i = 1;
     do while i le m;
     j = 1;
     do while j le n;

     { bstar,sse,xstar } = newb(rho[i,j]);  /*  gls       */

                                      /*  Equation 12.3.41 */

     store[i,j] = sqrt(1 - rho[i,j]^2)*(sse^(-nu/2))
                    / sqrt(det(xstar'xstar));
     j = j + 1;
     endo;
     i = i + 1;
     endo;

     retp(store);
endp;
```

With `RHOKERN` in memory, carry out the integral in (12.3.42) and solve for the normalizing constant.

```
_intord = 20;
xl = 1|-1;
```

```
rhoconst = intquad1(&RHOKERN,xl);
rhoconst = 1/rhoconst;
rhoconst;
```

Given the normalizing constant, write PROC `RHOPOST` which returns the p.d.f. values given ρ.

```
proc RHOPOST(rho);
     retp(rhoconst * RHOKERN(rho));
endp;
```

Graph the posterior p.d.f. for ρ as in Figure 12.5.

```
rhovec = seqa(0,.01,100);
library qgraph;
xy(rhovec,RHOPOST(rhovec));
```

The mean of the posterior distribution can be obtained using Equation 12.3.43. Again we will use numerical integration. Write PROC `RHOMU` which takes ρ as an argument and returns ρ times the p.d.f. value.

```
proc RHOMU(rho);
     local mom1;
     mom1 = rho .* RHOPOST(rho);
     retp(mom1);
endp;
```

Calculate the mean of the posterior distribution.

```
rhomean = intquad1(&RHOMU,xl);
rhomean;
```

To obtain the standard deviation of the posterior p.d.f. we will first calculate the second moment (about the origin) of the p.d.f. in the same fashion as the first moment.

```
proc RHOMU2(rho);
     local mom2;
     mom2 = rho .* rho .* RHOPOST(rho);
     retp(mom2);
endp;
```

Calculate the standard deviation of the posterior distribution.

```
rhose = sqrt(intquad1(&RHOMU2,xl) - rhomean^2);
rhose;
```

Calculate the probability that rho falls in the interval [0.4, 1.0].

```
xl = 1.0|.4;
p = intquad1(&RHOPOST,xl);
p;
```

The next task is to obtain the posterior distribution for an individual parameter value, β_2. The kernel of the joint posterior is given in Equation 12.3.44. It is a function of ρ and β_2. PROC RB2KERN takes these arguments and returns the value of the kernel. It must be written in such a way that it can be used by the bivariate numerical integration function INTQUAD2. In particular it must take arguments that are vectors or matrices and return the same.

```
proc RB2KERN(rho,b2);
     local m,n,p,q,store,i,j,bstar,sse,xstar,xtx,
                                ixx,c22,rhomat,b2m;

     store = 0 * rho .* b2;      /*  define storage matrix     */
     m = rows(store);            /*  obtain dimensions         */
     n = cols(store);

     rhomat = rho .* (store + 1);/*  matrix of rho values      */
     b2m = (store + 1) .* b2;    /*  matrix of b2 values       */

     i = 1;
     do while i le m;
     j = 1;
     do while j le n;
                                 /*  carry out gls             */

     { bstar,sse,xstar } = NEWB(rhomat[i,j]);

                                 /*  define pieces of kernel   */
     xtx = xstar'xstar;
     ixx = invpd(xtx);
     c22 = ixx[2,2];
                                 /*  calculate kernel          */

     store[i,j] = (sqrt(1 - rhomat[i,j]^2)*(sse^(-(nu + 1)/2))
                  /(sqrt(det(xtx))*c22))
                  /(1+((b2m[i,j]-1.672)^2)/(c22*sse))^((nu+1)/2) ;

     j = j + 1;
     endo;
     i = i + 1;
     endo;
     retp(store);
endp;
```

Now use INTQUAD2 to obtain the normalizing constant. Let the limits of β_2 be -1 to 4, which for practical purposes defines the real line. This will take a few minutes to calculate.

```
_intord = 20;
rhol = 1|-1;
b2l = 4|-1;
biconst = intquad2(&RB2KERN,rhol,b2l);
biconst;
```

Now write a procedure that calculates the value of the joint posterior as a function of ρ with β_2 in memory. The reason for this is that we will integrate out ρ, using INTQUAD1, to obtain the marginal posterior p.d.f. for β_2.

```
b2 = 0;

proc RB2POST(rho);
     local m,n,store,i,j,bstar,sse,xstar,xtx,ixx,c22,rhomat;
     store = 0 .* rho;
     m = rows(store);  n = cols(store);
     rhomat = rho .* (store + 1);
     i = 1;
     do while i le m;
     j = 1;
     do while j le n;
     { bstar,sse,xstar } = newb(rhomat[i,j]);
     xtx = xstar'xstar;
     ixx = invpd(xtx);
     c22 = ixx[2,2];
     store[i,j] = (1/biconst)
                  .* (sqrt(1- rhomat[i,j]^2)*(sse^(-(nu+1)/2))
                   / (sqrt(det(xtx))*c22))
                   / (1+((b2-1.672)^2)/(c22*sse))^((nu+1)/2);
     j = j+1;
     endo;
     i = i + 1;
     endo;

     retp(store);
endp;
```

Create a vector of values of beta2 and calculate the corresponding values of the posterior p.d.f.

```
b2vec = seqa(0,.035,100);
_intord = 10;
pdfvec = 0 * b2vec;
```

```
i = 1;
do while i le rows(b2vec);

b2 = b2vec[i];
pdfvec[i] = intquad1(&RB2POST,rhol);

i = i + 1;
endo;
```

Now graph the p.d.f. as in Figure 12.6.

```
xy(b2vec,pdfvec);
```

To calculate the mean and variance of the posterior p.d.f. we will use Equation 12.3.45. Write PROC MOMB2 which takes ρ as an argument and returns the values of the EGLS estimates of β_2 weighted by the posterior p.d.f. of ρ.

```
proc MOMB2(rho);
local m,n,store,rhomat,i,j,bstar,sse,xstar,ixx,c22;

m = rows(rho);
n = cols(rho);
store = zeros(m,n);
rhomat = rho .* (store + 1);
i = 1;
do while i le m;
j = 1;
do while j le n;
     {bstar,sse,xstar} = newb(rhomat[i,j]);
          store[i,j] = bstar[2] .* RHOPOST(rhomat[i,j]);
j = j + 1;
endo;
i = i + 1;
endo;
retp(store);
endp;

_intord = 20;
rhol = 1|-1;
meanb2 = intquad1(&MOMB2,rhol);
meanb2;


proc RB2POST2(rho,b2);
     local m,n,p,q,store,i,j,bstar,sse,
                              xstar,xtx,ixx,c22,rhomat,b2m;
```

```
    store = 0 * rho .* b2;        /*  define storage matrix   */
    m = rows(store);              /*  obtain dimensions       */
    n = cols(store);

    rhomat = rho .* (store + 1);  /*  matrix of rho values    */
    b2m = (store + 1) .* b2;      /*  matrix of b2 values     */

    i = 1;
    do while i le m;
    j = 1;
    do while j le n;
                                  /*  carry out gls           */

    { bstar,sse,xstar } = newb(rhomat[i,j]);

                                  /*  define pieces of kernel */
    xtx = xstar'xstar;
    ixx = invpd(xtx);
    c22 = ixx[2,2];
                                  /*  calculate kernel        */

    store[i,j] = (sqrt(1 - rhomat[i,j]^2)*(sse^(-(nu + 1)/2))
                 /(sqrt(det(xtx))*c22))
                 /(1+((b2m[i,j]-1.672)^2)/(c22*sse))^((nu+1)/2);

    j = j + 1;
    endo;
    i = i + 1;
    endo;
    retp(store./biconst);
endp;

proc MUB2(rho,b2);
    local mom1;
     mom1 = b2 .* RB2POST2(rho,b2);
     retp(mom1);
endp;

proc MU2B2(rho,b2);
    local mom2;
     mom2 = b2 .* b2 .* RB2POST2(rho,b2);
     retp(mom2);
endp;

_intord = 20;
rhol = 1|-1;
```

```
b2l = 4|-1;
meanb2 = intquad2(&MUB2,rhol,b2l);
meanb2;

mom2 = intquad2(&MU2B2,rhol,b2l);
mom2;

varb2 = mom2 - meanb2^2;
varb2;
seb2 = sqrt(varb2);
seb2;
```

The second example given in Section 12.3 is that of the model of multiplicative heteroskedasticity first presented in Section 9.3.4.
LOAD the data in file TABLE9.1 and check.

```
load dat[20,5] = table9.1;
y = dat[.,1];
x = ones(20,1)~dat[.,2 3];
z = x[.,1 2];
format 8,4;
y~x~z;
```

To obtain the ML estimates we can once again use PROC `MAXL`. First write a proc that returns the components of the log-likelihood function (12.3.48) given a vector of parameter values for beta and alpha stacked into a vector.

```
/*************************************************************/
/*                                                           */
/* PROC HETEROLI -- Log likelihood for multiplicative        */
/*                  heteroskedastic errors                   */
/*************************************************************/

proc HETEROLI(param0);
     local k,b,alpha,za,e,const;

 /* Take apart the parameter vector */

     k = cols(x);
     b = param0[1:k,.];
     alpha = param0[k+1:rows(param0),.];

 /* Compute the log likelihood   */

     za = z*alpha;
     e = y - x*b;
     const = -ln(2*pi)/2;
```

```
     retp(const - za/2 - exp(-za).*e.*e/2 );
endp;
```

Once the ML estimates are obtain the asymptotic standard errors can be obtained from the inverse of the information matrix, which is given in Equation (12.3.49) Write a proc to obtain those standard errors and put the covariance matrices into global memory for later use.

```
proc (2) = HETERCOV(param0);
     local k,b,alpha,xstar,za,covb,covalpha;

               k = cols(x);
               b = param0[1:k,.];
               alpha = param0[k+1:rows(param0),.];
               za = z*alpha;
               xstar = x .* (ones(1,k) .*. sqrt(exp(-za)));
               covb = invpd(xstar'xstar);
               covalpha = 2*invpd(z'z);

               "Final results  ";?;
               "Est.          "  param0';
               "Std. Errs.    "  sqrt(diag(covb))';;
                                 sqrt(diag(covalpha))';
       retp(covb,covalpha);
endp;
```

As suggested on p. 540 of ITPE2 let the starting values be the EGLS estimates.

```
          let param0 = 1.01 1.657 .896 -4.376 .366;
          param = MAXL(param0,&HETEROLI,x,y);
          {covb,covalpha} = HETERCOV(param);
```

As an alternative to using a general optimization algorithm for ML estimation it is sometimes possible to use the structure of the problem at hand to simplify matters. This is true for the model under consideration as was noted by Harvey. Equations (12.3.51) and (12.3.52) define iterations for the Method of Scoring algorithm. Note the convergence criteria.

```
/*************************************************************/
/*                                                           */
/*  PROC HARVEY -- ML estimation, by Method of Scoring,      */
/*                 of Multiplicative heteroskedasticity model */
/*************************************************************/

proc HARVEY(param0);
     local b0,a0,bn,an,iter,crit,e,za,estar,xstar,q;
```

```
    /*  Take apart initial parameter values   */

    k = cols(x);
    b0 = param0[1:k,.];
    a0 = param0[k+1:rows(param0),.];

    /*  Set initial constants   */

    bn = b0;
    an = a0;
    iter = 1;
    crit = 1;

    /*  Start do-loop and print  */

    do until (crit < 1e-8) or (iter > 50);
    " iter " iter " bn " bn' " an " an' " crit " crit;

    e = y - x*bn;          /*  transform residuals and X  */
    za = z*an;
    estar = e .* sqrt(exp(-za));
    xstar = x .* (ones(1,k) .*. sqrt(exp(-za)));

    bn = b0 + estar/xstar;         /*  Equation 12.3.51   */

    q = (exp(-za) .* e^2) -1;      /*  Equation 12.3.52   */
    an = a0 + q/z;

    /* check convergence  */

    crit = maxc(abs((bn|an) - (b0|a0)));
    b0 = bn;
    a0 = an;
    iter = iter + 1;
    endo;

    retp(bn|an);
endp;
```

Use this proc with the same starting values to obtain ML estimates.

```
let param0 = 1.01 1.657 .896  -4.346 .366;
param = HARVEY(param0)
{covb,covalpha} = HETERCOV(param);
```

In Section 12.3.4 asymptotic tests related to ML estimation are presented. They are applied to the model of multiplicative heteroskedasticity in Section 12.3.4b.

The Wald statistic in Equation (12.3.93) is simply the square of the asmptotic-t statistic in this case.

```
wald = (param[5,1] / sqrt( covalpha[2,2] ) )^2;
wald; cdfchic(wald,1);
```

The likelihood ratio test is given in Equation (12.3.102).

```
t = rows(x);
b = y/x;
sig02 = (y - x*b)'(y - x*b)/t;
lr = t*ln(sig02) - sumc(z*param[4 5,.]);
lr;
```

12.4 Nonlinear Seemingly Unrelated Regression Equations

In this Section the nonlinear SUR model is considered and maximum likelihood estimation of the concentrated likelihood function, Equation 12.4.9, described. As an example of such a model a linear expenditure system. The data used is that in file TABLE11.3 which contains T = 30 observations on the prices of 3 commodities, income and quantities of the commodities. LOAD the data and check it.

```
load dat[30,7] = table11.3;
p = dat[.,1:3];
y = dat[.,4];
q = dat[.,5:7];
v = p.*q;
format 10,5;
p~y~q;
```

In order to maximize the concentrated likelihood function we will use the Newton-Raphson algorithm, as described in Equation 12.2.89. The matrix of second partial derivatives will be approximated with numerical second derivatives using the **GAUSS** function HESSP, and numerical first derivatives using GRADP. The arguments of PROC MAXM are a set of initial estimates, b0, and PROC SURLI that defines the value (a scalar) of the objective function. This proc is long so place it in a separate file and run it. Note that this proc assumes that X and y are in memory. In other respects PROC MAXM is much like PROC `MAXL`.

```
/*****************************************************************/
/*                                                               */
/* MAXM.PRC -- Maximum likelihood estimation using the           */
/*             Newton-Raphson Algorithm (Data in Memory)         */
/*****************************************************************/

proc MAXM(b0,&ofn);

  local ofn:proc;
  local t,std,tol,H,g,b,db,iter,ofn1,ofn2,z,s,converge;

  db = 1;
  iter = 1;
  converge = 0;
  tol = 1e-5;

  do until converge == 1;

      g = gradp(&ofn,b0);
      H = hessp(&ofn,b0);
      db = -inv(H)*g';        /* -solpd(g,m) is much faster */
      gosub step;
      b = b0 + s*db;

      gosub prnt;

      db = abs(b-b0);
      b0 = b;

      if abs(db) < tol; converge = 1;
        else; iter = iter + 1;
      endif;
endo;

?;
"Final Results: ";
"  Coefficients: " b0';

std = sqrt(diag(-inv(H)));
"  Std. Errors:  " std';

retp(b0);

step:
  s = 2;
  ofn1 = 0;
```

```
  ofn2 = 1;
  do until ofn1 >= ofn2;
    s = s/2;
    ofn1 = ofn(b0 + s*db);
    ofn2 = ofn(b0 + s*db/2);
  endo;
return;

prnt:
  format 4,2; " i = " iter;;
  format 4,2; " Steplength = " s;;
  format 10,6; " Likelihood = " ofn(b0);
  format 10,6; " b = " b0';
  ?;
return;

endp;
```

Write a PROC that produces the value of the concentrated likelihood function for the first two of the equations in (12.4.13) (since the 3 equations are linearly dependent.)

```
proc surli(b0);
  local g,b1,b2,z,e1,e2,s1,s2,s12,s;

   g = b0[1:3,.];                /* Separate the param values  */
   b1 = b0[4,.];
   b2 = b0[5,.];

   z = y - p*g;                    /* Calculate Supernumerary
                                          income               */

   e1 = v[.,1] - p[.,1]*g[1,1] - b1*z;  /* Residuals           */
   e2 = v[.,2] - p[.,2]*g[2,1] - b2*z;

                                   /*  Elements of Contemporaneous
                                         Covariance            */
   s1 = e1'e1;
   s2 = e2'e2;
   s12 = e1'e2;
   s = (s1~s12)|(s12~s2);

  retp( -(rows(y)/2)*ln(det(s)) );  /*   Equation 12.4.9       */
endp;
```

Using the initial values suggested on p. 555 of ITPE2, obtain the ML parameter estimates. Note that the standard errors returned by `MAXM` are based on the

numerical second derivaties.

```
let b0 = 2.903 1.360 13.251 0.20267 .13429;
b = MAXM(b0,&surli);
```

12.5 Functional Form – The Box-Cox Transformation

In this Section maxmimum likelihood estimation of the regression and transformation parameters of the Box-Cox model is discussed. What follows is a series of PROCs that implement these ideas. The first is PROC `BCT` which carries out the transformation in Equation (12.5.3) of the text, given the matrix `Z` containing the data to transform and the parameter `lambda` which is either a scalar or a vector which is conformable to `z`.

```
/* PROC BCT -- Computing the Box-Cox Transformation */

proc BCT(z,lambda);
     local z1,z2,idx,zbc;
     idx = lambda .== 0;
     z1 = ((z^lambda) - 1)./lambda;
     z2 = ln(abs(z));
     zbc = (z2 .* idx) + (z1 .* (1-idx));
     retp(zbc);
endp;
```

Next, is PROC `BOXCOX`. Its arguments are a set of initial parameter values for β, λ and σ^2, in that order. PROC `BOXCOX` returns the T components of the log-likelihood function in Equation (12.5.10) in a (T x 1) vector. It assumes that y and X are in memory as well as a (T x 1) vector of ones, j. Thus, before proceeding, LOAD the data in TABLE12.7 and check it.

```
load dat[40,3] = table12.7;
x = dat[.,1 2];
y = dat[.,3];
j = ones(40,1);
x~y;
```

Now enter PROC `BOXCOX` and run it.

```
/* BOXCOX -- the full log-likelihood function */

proc BOXCOX(p0);
     local c,e,b,li,lambda,sigmasq;

/* Take apart the parameter vector */
```

```
     b = p0[1:rows(p0)-2,1];
     lambda = p0[rows(p0)-1,1];
     sigmasq = p0[rows(p0),1];

/* Compute the error term for the transformed data */

     e = BCT(y,lambda) - (j~BCT(x,lambda))*b;

/* Compute the log-likelihood */

     c = ln(2*pi*sigmasq);
     li = -(c/2) - (e.*e./(2*sigmasq)) + (lambda-1).*ln(y) ;

     retp(li);
endp;
```

Assume that the transformation parameter lambda is the same for all the variables. Use the starting values given at the top of p. 560 and the BHHH algorithm `MAXL` to obtain ML estimates. Make sure that `MAXL` is in memory.

```
let b0 = 3 1 1 1 1.5;
b = MAXL(b0,&BOXCOX,x,y);
```

Compare these estimates to those in Table 12.8 which assume the same lambda. The standard errors reported by `MAXL` are based on the BHHH approximation to the Hessian and using only first derivatives. Thus they are an approximation to the "Unconditional Standard Errors" reported in the text and, as you can see, not terribly close. ITPE2 cites the text by Fomby, Hill and Johnson, which contains a discussion of the difference between conditional and unconditional standard errors. The conditional standard errors assume that the transformation parameter λ is known, and not estimated, and are thus based on the covariance matrix in Equation (12.5.15) for β. Compute these conditional standard errors for the estimates of β based on the ML estimates.

```
xlam = j~BCT(x,.779);
covb = 1.24 * invpd(xlam'xlam);
std  = sqrt(diag(covb));
std';
```

The conditional standard error reported for σ^2 is based on its usual ML estimate of the variance.

```
varsig2 = 2*(1.24^2)/40;
std = sqrt(varsig2);
std;
```

In order to generalize the procedure to allow different λs on each of the variables enter the following procedure into a file and run it. It calculates the T elements of the Concentrated likelihood function in Equation (12.5.19). The PROC takes as argment a vector, `lam0`, that contains initial values for the transformation parameters for y and the (K - 1) regressors in X, in that order. Once again, it is assumed that `y`, `x`, and `j` are in memory. Note that it places into global memory OLS estimates `b` and `sigmasq` as well as the transformed X matrix which will be used in the next step using the **GAUSS** command CLEARG.

```
/*  PROC BOXCOX2 -- general Box-Cox Concentrated Likelihood    */

proc BOXCOX2(lam0);
     local dat,ybc,c,e,li;
     clearg b,xbc,sigmasq;

     /* Transform the data with the current lambda's */

     dat = BCT(y~x,lam0');
     ybc = dat[.,1];
     xbc = j~dat[.,2:cols(dat)];

     /* Compute the estimate of b (Equation 12.5.11) */

     b = ybc/xbc;

     /* Compute the estimate of sigamsq (Equation 12.5.12) */

     e = ybc - xbc*b;
     sigmasq = e'e/rows(y);

     /* Compute the components of the log-likelihood
                              in (Equation 12.5.19)       */

     c = ln(2*pi*sigmasq);
     li = -(c/2) - (e.*e/(2*sigmasq)) + (lam0[1,.]-1).*ln(y) ;
     retp(li);
endp;
```

Given the initial values of λ and estimates for β and σ^2, the maximum likelihood estimates of λ can be obtain and then the ML estimates of β and σ^2. However, so that approximate unconditional standard errors for all parameters can be obtained the proc below, `BOXCOX3`, obtains the full likelihood for this model as given exactly in Equation 12.5.19 given a set of parameters `p0` that contains estimates of β, λ, and σ^2, in that order.

```
/* BOXCOX3 -- Full likelihood with differing lambdas */
```

```
proc BOXCOX3(p0);
     local k,b,lam,sigmasq,dat,ybc,xbc,e,c,li;

     /* Take apart the parameter vector */

      k = cols(x) + 1;
      b = p0[1:k,1];
      lam = p0[k+1:2*k,1];
      sigmasq = p0[rows(p0),1];

     /* Transform the data */

      dat = BCT(y~x,lam');
      ybc = dat[.,1];
      xbc = j~dat[.,2:cols(dat)];

     /* Compute the likelihood */

      e = ybc - xbc*b;
      c = ln(2*pi*sigmasq);
      li = -(c/2) - (e.*e/(2*sigmasq)) + (lam[1,.]-1) .* ln(y);
     retp(li);
endp;
```

Finally we are ready to put it all together. Our strategy will be to use `MAXL` to obtain ML estimates of λ given some initial estimates. Then these ML estimates and the corresponding ML estimates for β and σ^2 will used to calculate the value of the full likelihood function which is fed into GRADP. Given the numerical values of the first derivatives the BHHH approximation to the Hessian is computed and approximate unconditional standard errors computed. All these steps are summarized in PROC `BOXFULL`. It takes as argument only the initial values for λ that `BOXCOX2` uses.

```
/* BOXFULL -- Combine BOXCOX2 and BOXCOX3 for full estimation */

proc BOXFULL(lam0);
     clearg lam,bvec,sigmasq,z,H,std,p;

/* Compute the vector of lambda's using BOXCOX2 */

     lam = MAXL(lam0,&BOXCOX2,x,y);

/* Create a complete parameter vector from values in memory */

     p = b|lam|sigmasq;
```

```
/* Compute the numerical gradients using gradp and BOXCOX3 */

     z = gradp(&BOXCOX3,p);

/* Compute the BHHH approximation to the Hessian */

     H = z'z;

/* Compute standard errors */

     std = sqrt(diag(invpd(H)));

/* Print out results */

     ?;
     " Estimates for Complete Model with Unconditional Std. Errors:";
     "   (b, lambda, sigmasq) ";
     p';
     std';
     retp(p);
endp;
```

Make sure all the procs are loaded into memory. Set the initial values of the parameters lambda to one and estimate the full model.

```
let lam0 = 1 1 1;
p = boxfull(lam0);
```

As you can see the numerical approximations to the unconditional standard errors are different from those in the text. To obtain the conditional standard errors we proceed as before.

```
xlam = j~BCT(x,-1.536~.391);
covb = 2.876 * invpd(xlam'xlam);
std  = sqrt(diag(covb));
std';


varsig2 = 2*(2.876^2)/40;
std = sqrt(varsig2);
std;
```

Chapter 13

Stochastic Regressors

In this chapter the consequences of not having fixed regressors are examined and the instrumental variable estimation technique used.

13.1 Independent Stochastic Regressor Model

When the regressor matrix is stochastic but independent of the error term then the least squares estimator has desirable properties and is equal to the ML estimator if the errors are normal.

13.2 Partially Independent Stochastic Regressors

If the regressors are only partially dependent on the errors, and not contemporaneously correlated with them then the usual properties of the least squares estimator hold asymptotically.

13.3 General Stochastic Regressor Models

When the errors are contemporaneously correlated with the error term the the usual least squares estimator is biased and inconsistent. Instrumental variable estimation is consistent but not asymptotically efficient, in general.
In Section 13.3.2 a numerical example is given. LOAD the data from TABLE13.1 and check it. It consists of 20 artificial observations on x_2, y and z_2.

```
load dat[20,3] = table13.1;
format 8,4;
dat;
```

Construct the matrices X and Z by adding an intercept variable.

```
t = rows(dat);
x = ones(t,1)~dat[.,1];
y = dat[.,2];
z = ones(t,1)~dat[.,3];
```

Compute the Instrumental Variables (IV) estimator in (13.3.24) and compare to the OLS estimates.

```
biv = x'y/x'z;
biv';

bols = y/z;
bols';
```

The asymptotic covariance matrix of the IV estimator is computed in Equation 13.3.25. First calculate the residuals and then the estimated variance, without correcting for the degrees of freedom.

```
ehat = y - z*biv;
sig2 = ehat'ehat/t;
sig2;
```

Compute the estimate of the asymptotic covariance matrix.

```
covb = sig2 * inv(x'z)*x'x*inv(z'x);
covb;
```

13.4 Measurement Errors

When errors of measurement exist in the X matrix a particular form of Errors in Variables or Stochastic Regressor model is created. In this section several estimators designed to deal with these problems are presented. A numerical example is given in Section 13.4.3 First LOAD the data in TABLE13.2. It consists of T = 15 observations on explanatory variables x_2 and x_3 and random disturbances u and v. Examine the data.

```
load dat[15,4] = table13.2;
t = rows(dat);
x = ones(t,1)~dat[.,1 2];
u = dat[.,3];
v = dat[.,4];
format 10,5;
x~u~v;
```

The random disturbances u and v shown in Table 13.2 are rounded values. Actually u and v are N(0, 0.2) and N(0, 0.5) and based on the first 30 "official" normal random numbers. We will construct the exact values so the solutions in the text can be reproduced exactly.

```
open f1 = nrandom.dat;
u = readr(f1,15);
v = readr(f1,15);
f1 = close(f1);
u = sqrt(.2)*u;
v = sqrt(.5)*v;
u~v;
```

Create the variables `zstar`, `z` and `y` using Equations (13.4.51)-(13.4.53).

```
let theta = 2 3 5;
zstar = x*theta;
z = zstar + u;
let beta = 10 0.8;
y = (ones(t,1)~zstar)*beta + v;
zstar~z~y;
```

Regress `z` on `x` to produce (13.4.54).

```
bz = z/x;
bz';
```

Form the predicted value of `z` and regress `y` on this predicted value, with an intercept, to produce (13.4.55)

```
zhat = x*bz;
binf = y/(ones(t,1)~zhat);
binf';
```

Alternatively, this estimate can be obtained directly from Equation 13.4.28.

```
zmat = ones(15,1)~z;
xz = x'zmat;
xty = x'y;
ixx = invpd(x'x);
binf = (xz'*ixx*xty)/(xz'*ixx*xz);
binf';
```

To calculate the second two-stage estimator regress `y` on `x` and compute the predicted value of `y`, as in (13.4.58)

```
by = y/x;
by';
yhat = x*by;
```

Then regress `z` on the predicted value of `y`, with an intercept, following Equation 13.5.49.

```
b0 = z/(ones(15,1)~yhat);
b0';
beta = 1/(b0[2,1]);
alpha = -b0[1,1]*beta;
alpha beta;
```

Or obtain the estimates directly from Equation 13.4.34.

```
xya = x'(ones(15,1)~y);
b0 = (xya'ixx*xty)/(xya'ixx*xz);
b0';
```

The third estimator is formed following (13.4.60)-(13.4.62),

```
syy = (y - yhat)'(y - yhat)/t;   /* Equation 13.4.38 */
syy;
szz = (z - zhat)'(z - zhat)/t;   /* Equation 13.4.39 */
szz;
lambda = syy/szz;                /* Equation 13.4.37 */
lambda;
```

The computation of the estimator (13.4.36b) actually assumes that the data is in deviation from the mean form (See comments below equations (13.4.25 and 13.4.26)) so correct for the mean values at this time.

```
yd = yhat - meanc(yhat);
zd = zhat - meanc(zhat);
yy = yd'yd;
zz = zd'zd;
zy = zd'yd;
blam = (yy - lambda*zz + sqrt((lambda*zz - yy)^2
       + 4*lambda*zy^2 ))/(2*zy);
blam';
```

To obtain Goldberger's Maximum Likelihood estimator the log-likelihood function in Equation 13.4.44 must be maximized. The following PROC `STOCHLI` computes the value of the log-likelihood function assuming `y`, `x` and `Z` are in memory. The argument is a vector of initial estimates of Π_z, as given in Equation 13.4.27, and consistent estimates of α and β in Equation (13.4.20), which are found by regressing y on $\hat{z}$, and an intercept, as in Equation 13.4.28. The proc returns the value of the log-likelihood, which can be maximized using PROC `MAXM`, which appears on page 133.

```
proc STOCHLI(p0);
     local li,k,t,k1,piz,piy,ey,ez,c;
     clearg sigv,sigu;
```

```
    k = rows(p0);
    t = rows(x);
    piz = p0[1:k-2,.];
    alpha = p0[k-1,.];
    beta = p0[k,.];
    piy = beta*piz;
    piy[1,.] = piy[1,.] + alpha;
    ey = y - x*piy;
    ez = z - x*piz;
    sigv = ey'ey/t;
    sigu = ez'ez/t;
    c = -t*ln(2*pi) - t*.5*ln(sigu) - t*.5*ln(sigv);
    li = c - (ey'ey)/(2*sigv) - (ez'ez)/(2*sigu) ;
    retp(li);
endp;
```

Run the procedure and make sure `MAXM` is in memory. Then, obtain initial estimates as suggested above and form the vector `p0` of initial values.

```
bz = z/x;
zhat = x*bz;
binf = y/(ones(t,1)~zhat);
p0 = bz|binf;
```

Minimize the objective funtion to obtain ML estimates of the parameters.

```
p = MAXM(p0,&STOCHLI);
```

Chapter 14

Simultaneous Linear Statistical Models: I

14.1 Introduction

In this chapter the nature of simultaneous equations models is explored. In particular the notation and assumptions are developed, the inconsistency of OLS estimation demonstrated and the concept of identification explained.

14.2 Specification of the Sampling Model

In this Section the notation is developed and the assumptions of the sampling model stated. Following Equation 14.2.21 a numerical example is given. Define the matrices sigma and plimx as given at the top of page 608.

```
let sigma[2,2] = 5 1
                 1 1;

let plimx[3,3] = 1  1  0
                 1  2  0
                 0  0  1;
```

Define the parameter matrices Γ and B.

```
let g[2,2]     = -1  2
                    1 -1;

let b[3,2]     =  0  3
                  2  0
                  0  1;
```

Compute reduced form parameters as in Equation 14.2.13a.

```
format 8,4;
pix = -b*inv(g);
pix;
```

Compute the plim of $V'V/T$, which is the contemporaneous covariance matrix for the reduced form disturbances. See Equation 14.2.18a.

```
plimv = inv(g)'sigma*inv(g);
plimv;
```

Compute the plim of $y'y/T$. See Equation 14.2.23. The (2,2) element should agree with (14.2.23).

```
plimy = pix'plimx*pix + plimv;
plimy;
```

Compute the plim of $X'y/T$. See Equation 14.2.24. The (2,2) element should agree with (14.2.24).

```
plimxy = pix'plimx;
plimxy;
```

Compute the plim of $y'e/T$. See Equation 14.2.25. The (2,1) element will agree with (14.2.25).

```
plimye = -inv(g)'sigma;
plimye;
```

14.3 Least Squares Bias

In this Section the least squares bias and inconsistency is demonstrated. On page 611 the example from the preceeding section is continued.
Compute the plim of the least squares estimator as in Equation 14.3.7. Construct `plimz` from the previous results.

```
let plimz[2,2] = 91  -11
                -11    2;
```

Define δ_1.

```
let d1 = 1 2;
```

Construct `plimze` using previous results.

```
let plimze = -11 0;
```

Find the plim of the least squares estimator of delta and its asymptotic bias.

```
plimd1 = d1 + inv(plimz)*plimze;
format 8,4;
plimd1;
bias = plimd1 - d1;
bias;
```

14.4 The Problem of Going from the Reduced-Form Parameters to the Structural Paramaters

In this Section the Identification problem is defined and identification rules given. In Section 14.5.3 an Empirical example is given using the simple Keynesian model.
LOAD the data from file TABLE14.1. It consists of T = 20 observations on investment from Table 14.1 in ITPE2.

```
load i[20,1] = table14.1;
```

Using i, and the official random numbers, we can create the remaining elements of the model using Equations 14.9.1-14.9.2. Let e be a vector of N(0,.04) random disturbances.

```
open f1 = nrandom.dat;
e = readr(f1,20);
f1 = close(f1);
e = sqrt(.04)*e;
```

Define the parameter values for α and β.

```
alpha = 2;
beta = 0.8;
```

Construct v.

```
v = (1/(1 - beta))*e;
```

Construct c and y using Equation 14.9.2.

```
c = alpha/(1 - beta) + beta/(1 - beta) * i + v;
y = alpha/(1 - beta) + 1/(1 - beta) * i + v;
```

Print out Table 14.1 using the constructed values.

```
i~c~y~v;
```

Compute reduced form parameters regressing c and y on i.

```
dep = c~y;
x = ones(20,1)~i;
pix = dep/x;
pix;
```

Solve for the the structural parameters using (14.5.23) and (14.5.25).

```
pi11 = pix[1,1];
pi21 = pix[2,1];
beta = pi21/(1+pi21);
alpha = pi11*(1 - beta);
alpha beta;
```

Then, using income equations, (14.5.24) and (14.5.26),

```
pi12 = pix[1,2];
pi22 = pix[2,2];
beta = (pi22 - 1)/pi22;
alpha = pi12*(1 - beta);
alpha beta;
```

Compare these results to the OLS estimates.

```
b = y/(ones(20,1)~c);
b';
```

Since it is impossible to judge the amount of estimator bias from one sample of data, carry out a short monte carlo experiment, repeating the above using n = 250 samples of size T = 20.
Create the data on c and y.

```
n = 250;
t = 20;
load e = e1nor.fmt;
e = sqrt(.04) * e;
c = (2 + .8*i + e)/(1 - .8);
y = i + c;
```

Obtain reduced form estimates and calculate their mean values from the 250 samples and compare to the true values for the consumption equation (14.5.25).

```
x = ones(t,1)~i;
pic = c/x;
meanc(pic')';
```

Obtain estimates of the structural parameters for the 250 samples and compare their mean to the true structural parameter values.

```
p1 = pic[1,.];
p2 = pic[2,.];
beta = p2 ./ (1+p2);
alpha = p1 .* (1 - beta);
b = alpha|beta;
?;
meanc(b')';
stdc(b')';
```

Obtain the OLS estimates for the consumption equation for 250 samples and calulate their mean and standard deviation.

```
j = 1;
bols = zeros(2,n);
do until j > n;

       bols[.,j] = c[.,j]/(ones(t,1)~y[.,j]);
       j = j + 1;

endo;
meanc(bols')';
stdc(bols')';
```

Calculate the percent of samples in which the OLS estimates were greater than the indirect least squares estimates.

```
z = (bols - b) .> 0;
meanc(z');
```

Calculate the percent of samples in which the indirect least squares estimates were greater than the true parameter values.

```
let beta = 2 .8;
z = b .> beta;
meanc(z');
```

Calculate the percent of samples in which the OLS estimates were greater than the true parameter values.

```
z = bols .> beta;
meanc(z');
```

These results should demonstrate to you the extent of the OLS bias. You may wish to experiment with larger sample sizes.

Chapter 15

Simultaneous Linear Statistical Models: II

In this chapter a variety of estimators for single equations within a system of simultaneous equations are considered as well as the problem of estimating all the equations jointly. The asymptotic properties of the estimators are determined and compared.

15.1 Estimating the Parameters of an Overidentified Equation

When equations within a simultaneous system are overidentified the indirect least squares estimator is inefficient. Generalized and Two Stage Least Squares estimators are proposed that are more efficient than the indirect least squares estimator.

15.2 The Search for an Asymptotically Efficient Estimator

The estimators proposed in Section 15.2 are single equation methods. That is they estimate the parameters of a single structural equation at a time. Much as in the case in Seemingly Unrelated Regression problems the efficiency of estimation can be increased if contemporaneous correlations exist among the structural equation errors and if the equations are overidentified. The technique of Three Stage Least Squares is introduced as a method of using this additional information.

15.3 Asymptotic and Finite Sampling Properties of the Alternative Estimators

This Section summarizes the properties of the various estimators.

15.4 An Example

To illustrate the various estimators discussed in the previous Sections an example is carried out. First we will verify the sample generation process by replicating the data generated in Table 15.1.
LOAD the data in file TABLE15.1, which consists of T = 20 observations on the five exogenous variables listed in Table 15.1. Check the data.

```
load x[20,5] = table15.1;
format 10,7;
x;
```

Specfify the Gamma and Beta matrices of structural parameters, given in Equations 15.4.3 and 15.4.4, and then construct the matrix of reduced form parameters in (15.4.6).

```
let gam[3,3] = -1  0.2  0
               -10  -1  2
               2.5   0 -1;

let beta[5,3] = -60 40 -10
                  0 -4  80
                  0 -6   0
                  0 1.5  0
                  0  0   5;

pimat = -beta*inv(gam);
pimat;
```

Specify the Sigma matrix in Equation 15.4.5.

```
let sigma[3,3] = 227.55    8.91   -56.89
                   8.91    0.66    -1.88
                 -56.89   -1.88    15.76;
```

In order to generate random disturbances with a N(0, Sigma) distribution we will follow the same process used in Chapter 11, as described in the Appendix to that chapter. First, find the characteristic roots and vectors of Σ and change their order to one of descending magnitude.

```
{d,c} = eigrs2(sigma);
d c;

d = rev(d);
c = (rev(c'))';
d c;
```

Due to the normalization issue of characteristic vectors, discussed in Chapter 11, the sign of the third characteristic vector must be changed for us to exactly replicate the data in ITPE2.

```
c[.,3] = -c[.,3];
c;
```

Create the transformation matrix and check that it does transform the matrix sigma to the identity and that when multiplied by its transpose it creates the sigma matrix.

```
sighalf = c * diagrv( eye(3), sqrt(d) );

check1 = c'*sigma*c;
check1;

check2 = sighalf*eye(3)*sighalf';
check2;
```

Read in the first 60 "official" N(0,1) random numbers.

```
open f1 = nrandom.dat;
e1 = readr(f1,20);
e2 = readr(f1,20);
e3 = readr(f1,20);
f1 = close(f1);
```

Stack the columns into a matrix E and transform them using `sighalf`. The resulting random numbers have the desired sampling distribution.

```
e = (e1~e2~e3)*sighalf';
```

Create the reduced form disturbances `v` and use the reduced form equation to create the values of y.

```
v = e*inv(gam);
y = x*pimat + v;
y;
```

Using the data on y and X, estimate the reduced form parameters (15.4.10).

```
pix = y/x;
pix;
```

Compute the OLS parameter estimates for each structural equation (15.4.11-15.4.12).

```
y1 = y[.,1];
z1 = y[.,2 3]~x[.,1];
b1 = y1/z1;

y2 = y[.,2];
z2 = y[.,1]~x[.,1 2 3 4];
b2 = y2/z2;

y3 = y[.,3];
z3 = y[.,2]~x[.,1 2 5];
b3 = y3/z3;

b1';
b2';
b3';
```

Since the second equation is just identified, its structural parameters can be efficiently estimated using indirect least squares (15.4.13).

```
bils = x'y2/x'z2;
bils';
```

For the overidentified equations indirect least squares is inefficient and GLS-2SLS should be used. See Equation 15.1.10.

```
q = x*inv(x'x)*x';

bgls1 = (z1'q*y1)/(z1'q*z1);        /*  Equation 1  */
bgls2 = (z2'q*y2)/(z2'q*z2);        /*  Equation 2  */
bgls3 = (z3'q*y3)/(z3'q*z3);        /*  Equation 3  */
```

To obtain standard errors for these estimators we will estimate the covariance matrix as in Equations 15.1.15 and 15.1.16, correcting for the degrees of freedom.

```
e1 = y1 - z1*bgls1;
e2 = y2 - z2*bgls2;
e3 = y3 - z3*bgls3;

t = rows(e1);
ehat = e1~e2~e3;
sse = diag(ehat'ehat);
```

```
k = cols(z1)|cols(z2)|cols(z3);
df = t - k;
var = sse ./ df;

sd1 = sqrt(diag( var[1] * invpd(z1'q*z1) ));
sd2 = sqrt(diag( var[2] * invpd(z2'q*z2) ));
sd3 = sqrt(diag( var[3] * invpd(z3'q*z3) ));
```

Print out the estimates and their standard errors and compare to (15.4.17).

```
format 8,4;

(bgls1~sd1)';
?;
(bgls2~sd2)';
?;
(bgls3~sd3)';
```

To compute the 3SLS estimator the contemporaneous covariance matrix will be estimated using the 2SLS residuals and the inverse taken.

```
sig = ehat'ehat/t;
isig = invpd(sig);
```

The most direct computational approach is to use Equation 15.2.4, which requires the construction of the stacked vector `y` and block diagonal matrix `z` as indicated below (15.2.2).

```
y = y1|y2|y3;

k1 = cols(z1);
z1a = z1|zeros(2*t,k1);

k2 = cols(z2);
z2a = zeros(t,k2)|z2|zeros(t,k2);

k3 = cols(z3);
z3a = zeros(2*t,k3)|z3;

z = z1a~z2a~z3a;
```

The 3SLS estimator is then computed. Compare these results to those in (15.4.21).

```
num = z'(isig.*.q)*y;
den = z'(isig.*.q)*z;
b3sls = num/den;
std = sqrt(diag(invpd(den)));
b3sls~std;
```

The only problem with this approach is that the matrices involved can be large and storage can become a problem. The following is equivalent code which does not involve creating the large matrices. Analyze how these statements work.

```
z = z1~z2~z3;
y = y1~y2~y3;
indx = ones(3,1)|( ones(5,1)*2 )|( ones(4,1)*3 );
vv = isig[indx,indx];
vy = isig[indx,.];
xx = z'x*invpd(x'x)*x'z;
zy = z'x*invpd(x'x)*x'y;
isxx = invpd(xx .* vv);
szy = sumc( (zy .* vy)' );
b = isxx*szy;
std = sqrt(diag(isxx));
b~std;
```

Finally, write a PROC computing the reduced form coefficients from a given set of structural estimates that are stacked into a single column, c.

```
proc RFORM(c);
   local g, b;

   g = -eye(3);                    /*  Gamma       */
   g[2 3,1] = c[1 2,1];            /*  Equation 1  */
   g[1,2]   = c[4,1];              /*  Equation 2  */
   g[2,3]   = c[9,1];              /*  Equation 3  */

   b = zeros(5,3);                 /*  Beta        */
   b[1,1] = c[3,1];                /*  Equation 1  */
   b[1:4,2] = c[5:8,1];            /*  Equation 2  */
   b[1 2 5,3] = c[10:12,1];        /*  Equation 3  */
   retp( -b*inv(g) );
endp;
```

Run PROC `RFORM` to put it into memory and then apply it to the OLS, GLS-2SLS and 3SLS parameter estimates.

```
bols  = b1|b2|b3;
pols = rform(bols);
pols;                           /*  Equation 15.4.22  */


b2sls = bgls1|bgls2|bgls3;
p2sls = rform(b2sls);
p2sls;                          /*  Equation 15.4.23  */
```

```
p3sls = rform(b3sls);
p3sls;                          /*  Equation 15.4.24  */
```

15.5 On Using the Results of Econometric Models for Forecasting and Decision Purposes

In this Section some of the uses of econometric models are discussed. In Equations 15.5.1a-g the use of these models as forecasting tools is illustrated. First, if x_t is given by (15.5.1e) and if the true reduced form parameters are known the forecasted value of y_t is (15.5.1f)

```
let xt = 1 8 6 23 40;
yt = xt'pimat;
```

If the reduced form parameters are derived from the 3SLS estimates the forecasted value of yt is (15.5.1g).

```
yhat = xt'p3sls;
yhat;
```

In the context of a dynamic model the characteristic roots of the matrix F containing the parameters on the lagged endogenous variables determine the dynamic properties of the model. For the system in Equations 15.5.6a-b the matrix F is given in (15.5.8). This matrix is not symmetric and the resulting characteristic roots need not be real. **GAUSS** can handle this problem, however, as it contains special functions designed for complex numbers. The **GAUSS** function EIGRG returns the real and imaginary parts of the characteristic roots of a real, general matrix.
Note that the values in the text are incorrect. This may be verified by noting that the determinant of F is the product of its characteristic roots.

```
let f[2,2] = .32 -.18 -.16 -.06;
{crr,cri} = eigrg(f);
crr~cri;
```

The lag 1-step dynamic multipliers are calculated from $F * G$ in (15.5.9).

```
let g[2,3] = 6.8 .04 .3
             1.6 -.02 .1;
f*g;
```

Chapter 16

Time-Series Analysis and Forecasting

16.1 Introduction

In this Chapter "pure" time series models are presented. Observations on a random variable are considered a realization from a stochastic process. Thus the purposes of this Chapter are to discuss ways to model stochastic processes, to estimate the parameters of such a model and to use the estimated model to forecast future values of the variable.

16.2 A Mathematical Model for Time-Series and Its Characteristics

In this Section stochastic processes are defined and definitions of the autocovariance and autocorrelation functions given. The concept of stationarity is discussed and lag operator notation defined.

16.3 Autoregressive Processes

Autoregressive processes use past values of a random variable to explain present and future values. To illustrate several data sets are examined. First an artificially constructed sample is considered.
LOAD the data in file TABLE16.1. It consists of 100 observations on a random variable y generated from an AR(2) process described on page 685 of ITPE2. Examine the data;

```
load y[100,1] = table16.1;
y';
```

Plot the data against time.

```
t = rows(y);
x = seqa(1,1,t);
library qgraph;
xy(x,y);
```

Write a proc that computes the partial autocorrelations for a data series, given the data series and the maximum lag to be considered. The partial autocorrelations are estimated by using the least squares estimator in (16.3.4) for model (16.3.3). The maximum lag included is increased sequentially and the corresponding coefficient is the partial autocorrelation coefficient for that lag value. The programming difficulty is that the number of complete observations changes as the lag length is increased. Place PROC `PARTIAL` in a convenient file and run it.

```
proc  PARTIAL(y,maxk);
local thetakk,t,k,yk,xk,thetak,tk;

y = y - meanc(y);              /*  center data                  */
thetakk = zeros(maxk,1);       /*  storage vector for estimates */
t = rows(y);
k = 1;                         /*  obtain estimate for 1st lag  */
yk = y[k+1:t,1];
xk = y[k:t-1,1];
thetak = yk/xk;
thetakk[k,1] = thetak;
k = 2;                          /*  begin loop                  */
do while k le maxk;
     yk = y[k+1:t,1];           /*  define yk                   */
     tk = t-k;                  /*  number of complete obs      */
     xk = xk[1:tk,.];           /*  delete last obs             */
     xk = y[k:t-1,1]~xk;        /*  add lag                     */
     thetak = yk/xk;            /*  OLS                         */
     thetakk[k,1] = thetak[k,1];/*  store coefficient           */
     k = k+1;
endo;
retp(thetakk);

endp;
```

Use PROC `PARTIAL` to estimate the partial autocorrelations for the data y and compare the results to Table 16.2 in ITPE2.

```
thetakk = partial(y,12);
thetakk';
```

The approximate 95% bounds are given in (16.3.12). If the absolute value of a partial autocorrelation coefficient is greater than $2/\sqrt{t}$ then it is deemed significantly different from zero.

```
bound = 2/sqrt(t);
bound;
```

On the basis of the sample partial autocorrelations we (correctly) identify the process as AR(2). The estimation results are

```
p = 2;
yp = y[p+1:t,1];
xp = y[p:t-1,1]~y[p-1:t-p,1];
thetap = yp/xp;
thetap';
sig2 = (yp - xp*thetap)'(yp - xp*thetap)/(t-2*p);
sig2;
```

Compare the resulting estimates to the true values.
A second data set is given in Table 16.3. It is artificial data generated from a MA(1) process. The file TABLE16.3 contains the 100 values of the actual, unobservable errors and the observable random variable y. LOAD the data and define y.

```
load dat[100,2] = table16.3;
t = rows(dat);
y = dat[.,2];
y';
```

Graph y against time.

```
x = seqa(1,1,t);
xy(x,y);
```

Obtain the partial autocorrelations for lags 1 - 15 and compare the the bound value.

```
thetakk = partial(y,15);
thetakk';
```

There are "significant" partial autocorrelations for relatively high lags.

16.4 Moving Average Processes

In this section moving average processes are defined. Identification of a MA processs involves the autocorrelation function. Two different estimators for autocorrelations are given in (16.4.9) and (16.4.11). Write a proc to obtain both of these estimators of autocorrelation given the data series and the maximum lag length. Place PROC `AUTOCORR` in a convenient file and run it.

```
proc (2) = autocorr(y,kmax);
local t,ybar,yd,c0,rk,rkbar,k,yt,ytk,ck,ckbar;

t = rows(y);
ybar = meanc(y);              /*   center data               */
yd = y-ybar;
c0 = yd'yd/t;                  /*  estimate variance - c0    */
rk = zeros(kmax,1);            /*  define storage matrices   */
rkbar = zeros(kmax,1);
k = 1;                         /*  begin loop                */
do while k le kmax;
     yt = yd[1:t-k,1];         /*  y in period t             */
     ytk = yd[1+k:t,1];        /*  y in period t+k           */
     ck = yt'ytk/t;            /*  Eq. 16.4.10               */
     ckbar = yt'ytk/(t-k);     /*  Eq. 16.4.12               */
     rk[k,1] = ck/c0;          /*  Eq. 16.4.9                */
     rkbar[k,1] = ckbar/c0;    /*  Eq. 16.4.11               */
     k = k+1;
endo;
retp(rk,rkbar);

endp;
```

Use PROC `AUTOCORR` to obtain the autocorrelation function for the data y from Table 16.3 for 15 lag periods. Compare the values to the bound value.

```
{rk,rkbar} = autocorr(y,15);
rk~rkbar;
```

Based on these autocorrelations the data series is identified to be a MA(1) process.
To estimate the parameters of a MA(q) process the sum of squares objective function in (16.4.18) must be minimized. In practice (16.4.21) is minimized. Write a proc to calculate this objective function given the data series y and a value of the single parameter, a. The ojective function is complicated by the fact that the number of terms in the summed quantity increases as the index of summation changes. Place PROC `MA1SUM` in a file and run it. In general, the data should be centered by subtracting the mean of the data prior to minimization. The solutions in the text assume that the data has zero mean and thus that centering is not required. For that reason PROC `MA1SUM` includes the centering step as a comment and is not used.

```
proc MA1SUM(a,y);
local t,obj,i,incr,k;

/*  y = y - meanc(y); */       /*  center the data           */
t = rows(y);
```

```
obj = 0;                          /*  initialize obj fn          */
i = 1;                            /*  begin loop to sum over t   */
do while i le t;
incr = 0;                         /*  initialize increment       */
k = 1;                            /*  begin loop for summand     */
do while k le i;
incr = incr +  y[k,1] .* a^(i-k); /* t'th term of Eq. 16.4.21  */
k = k+1;
endo;

obj = obj + incr^2;               /*  Eq. 16.4.21                */
i = i+1;
endo;
retp(obj);

endp;
```

It is possible to minimize this objective function with respect to a using a numerical optimization algorithm like in Chapter 12. It is tedious, however, as the objective funtion itself is cumbersome to evaluate. In this case it is simpler to use a numerical search to obtain the minimizing value. First search over a rough grid from -0.9 to 0.9 to find the minimizing value.

```
avec = seqa(-0.9,.1,9)|0|seqa(.1,.1,9);
obj = ma1sum(avec,y);
obj';

ahat = avec[minindc(obj),1];
ahat;
```

Then search over a finer grid of values near the minimizing value to obtain a final estimate.

```
avec = seqa(ahat-0.1,.01,20);
obj = ma1sum(avec,y);
ahat = avec[minindc(obj),1];
ahat;
```

Obtain estimates of the error variance.

```
obj = minc(obj);
sighat2 = obj/(t-1);
sigtil2 = obj/t;
sighat2 sigtil2;
```

16.5 ARIMA Models

In this section the features of AR and MA processes are combined. A model with both MA and AR terms is called an ARMA(p,q) model. If the data must be differenced to achieve stationarity it is called an ARIMA(p,d,q) process.

16.6 The Box-Jenkins Approach

The Box-Jenkins approach to time-series model building is given in this section. There are separate steps for identification, estimation and diagnostic checking. The example used is data on corn prices which is given in Table 16.4. LOAD file TABLE16.4 and examine the data.

```
load y[82,1] = table16.4;
y';
```

Plot the data against time.

```
t = rows(y);
x = seqa(1,1,t);
xy(x,y);
```

Compute the partial autocorrelations for 10 periods and compare them to the 2 standard deviation bound.

```
thetakk = partial(y,10);
thetakk';
bound = 2/sqrt(t);
bound;
```

Compute the autocorrelations for 15 periods.

```
{rk,rkbar} = autocorr(y,15);
rk~rkbar;
```

Based on these calculations the series is identified as an AR(1) process. Obtain the parameter estimates for model (16.6.1).

```
yt = y[2:t,1];
yl = y[1:t-1,1];
x = ones(t-1,1)~yl;
t = rows(x);
k = cols(x);
bhat = yt/x;
sighat2 = (yt - x*bhat)'(yt - x*bhat)/(t-k);
covb = sighat2 * invpd(x'x);
stderr = sqrt(diag(covb));
bhat';
stderr';
sighat2;
```

Using these values calculate an estimate of mu as in (16.6.3) and calculate its approximate asymptotic standard error. Your values will differ from those in the text, which are based on the "rounded" values in Equation 16.6.2.

```
nu = bhat[1,1];
rho = bhat[2,1];
mu = nu/(1-rho);
sdmu = sqrt(sighat2/((1-rho)^2 * t));
mu;
sdmu;
```

In order to check the AR(1) specification obtain the least squares residuals and calculate the autocorrelations. No distinct pattern should be present.

```
ehat = yt - x*bhat;
{rk,rkbar} = autocorr(ehat,15);
rk~rkbar;
```

Calculate the portmanteau test statistic Q in (16.6.4) as a further check. If the AR(1) process is correctly specified the statistic Q is approximately chi-square with K-1 = 14 degrees of freedom.

```
kvec = seqa(1,1,15);
q = t*(t+2)*sumc( rk^2 ./(t-kvec));
q;
cdfchic(q,15-1);
```

16.7 Forecasting

The estimated model can be used to generate forecasts of future values of the stochastic process. Use Equation 16.7.14 to forecast one-step ahead. Your values will differ from those in the text which uses the rounded parameter estimates in Equation 16.6.2.

```
let x1 = 1 1114;
yhat1 = x1'*bhat;
yhat1;
```

Now repeat the process, using the past forecasts to generate the next future value.

```
x2 = 1~yhat1;
yhat2 = x2*bhat;
x3 = 1~yhat2;
yhat3 = x3*bhat;
x4 = 1~yhat3;
yhat4 = x4*bhat;
x5 = 1~yhat4;
yhat5 = x5*bhat;
```

Store the forecasts in a vector for future use.

```
yhat = yhat1|yhat2|yhat3|yhat4|yhat5;
```

In order to construct forecast intervals for these values a forecasting mean square error must be constructed for each. As Equation 16.7.11 illustrates the forecast MSE depends on the coefficents of the MA representation. The i'th MA coefficient is shown to be $.8^i$ in the middle of page 711. Construct the first five of these terms.

```
phi0 = 1;
phi1 = .8;
phi2 = .8^2;
phi3 = .8^3;
phi4 = .8^4;
```

Use Equation 16.7.11 to construct forecast MSEs for 5 periods into the future.

```
sig1 = sighat2;
sig2 = sig1*(1 + phi1^2);
sig3 = sig1*(1 + phi1^2 + phi2^2);
sig4 = sig1*(1 + phi1^2 + phi2^2 + phi3^2);
sig5 = sig1*(1 + phi1^2 + phi2^2 + phi3^2 + phi4^2);
sigvec = sig1|sig2|sig3|sig4|sig5;
```

Obtain 95% confidence bounds and construct Table 16.5

```
lb = yhat - 1.96*sqrt(sigvec);
ub = yhat + 1.96*sqrt(sigvec);
yhat~sigvec~lb~ub;
```

Chapter 17

Distributed Lags

17.1 Introduction

When using time series data there is often a time lag between an event and its effect. Furthermore the impacts of an event may be spread over more than one future time period. Models that take these factors into account are called distributed lag models. In this chapter finite lag models and infinite lag models are considered. Finite lags reflect the assumption that the future effects of an event are exhausted in a specific, or finite, time period. Infinite lags assume that the effect of the event is spread over an infinite horizon.

17.2 Unrestricted Finite Distributed Lags

For finite distributed lags it is common practice to use the data to help select the length of the lag. In file TABLE17.1 is a data set on quarterly capital appropriations (x) and expenditures (y). LOAD the data and examine it.

```
load dat[88,2]= table17.1;
yvec = dat[.,1];
yvec';

xvec = dat[.,2];
xvec';
```

Note that observation 42 on y is different than in the text. We will use this data as it is the basis of the calulations in the text.
Following the example in the text, page 725, assume that the maximum lag length to be considered is M = 10. Construct the y vector, beginning with observation M + 1.

```
t = rows(xvec);
m = 10;
```

```
y = yvec[m+1:t,1];
```

In order to construct Table 17.2 in the text we will begin construction of the X matrix in Equation 17.2.4 assuming that the lag length is zero and then add columns as the lag length increases to its maximum of 10. The design matrix is initially a column vector of ones representing the equation intercept. As the lag length is increased calculate the statistics in Table 17.2 using PROC LAGSTAT, below, and store them in a storage matrix, store. Execute the procedure to place it in memory.

```
proc  LAGSTAT(y,x);
     local t,k,n,b,sse,sig2,aic,sic;
     t = rows(x);
     k = cols(x);
     n = k-2;                          /*  n is the lag length  */
     b = y/x;
     sse = (y - x*b)'(y - x*b);
     sig2 = sse/(t-k);                 /*  Eq. 17.2.5           */
     aic = ln(sse/t) + 2*n/t;          /*  Eq. 17.2.11          */
     sic = ln(sse/t) + n*ln(t)/t;      /*  Eq. 17.2.12          */
     retp(sse~sig2~aic~sic);
endp;
```

Now construct an X matrix consisting of a (T - M) x 1 column of ones and create the storage matrix.

```
x = ones(t-m,1);
store = zeros(m+1,5);
```

Write a DO-LOOP within which the values in Table 17.2 are calculated for lag lengths $n = 0, \ldots, M = 10$.

```
n = 0;
do while n le m;
x = x~xvec[m-n+1:t-n,1];
store[n+1,.] =n~lagstat(y,x);
n = n + 1;
endo;
```

Print the matrix store and compare to Table 17.2.

```
     format 14,9;
"--------n-------------sse----------sighat2---------aic----------sic";
     store;
```

Calculate the sequential test statistics given in Equation 17.2.7 and used as a basis for determining the lag length. Carry out each test at the .05 level of significance and stop the testing process once a hypothesis is rejected. The lag length is the last value tested but not rejected as zero.

```
sse = store[2:11,2];
sighat2 = store[2:11,3];

test = 1;
pval = 1;

do until pval le .05;
     n = m - test;
     lam = (sse[n,1] - sse[n+1,1])/sighat2[n+1,1];
     pval = cdffc(lam,1,t-(n+2));
     n~lam~pval;
     test = test + 1;
endo;
```

Compare these results to those on page 725. On the basis of these tests the lag length is chosen to be N = 8. Given this lag length, obtain the OLS parameter estimates of the distributed lag weights, correcting the sample size to T - N.

```
nhat = 8;                              /*  lag length    */

t = rows(yvec);
y = yvec[nhat+1:t,1];                  /*  specify y     */

x = ones(t-nhat,1);                    /*  construct X   */
n = 0;
do while n le nhat;
     x = x~xvec[nhat-n+1:t-n,1];
     n = n + 1;
endo;

k = cols(x);
t = rows(x);

b = y/x;                               /*  OLS           */
sighat2 = (y-x*b)'(y-x*b)/(t - k);
covb = sighat2*invpd(x'x);
stderr = sqrt(diag(covb));
b~stderr;
```

In comparing these results to those in Equation 17.2.9 note that there are some differences in the estimates. This is due to roundoff error resulting from the highly multicollinear nature of the explanatoray variables in this regression. The topic of multicollinearity is explored in Chapter 21.

17.3 Finite Polynomial Lags

One way to deal with the inherent multicollinearity in the finite distributed lag model is to impose some structure on the lag weight distribution. A popular and easily implemented alternative is to assume that the lag weights fall on a polynomial of some low degree, implying that the lag weight distribution is "smooth".
To extend the example in the previous section, let the lag length be specified as eight. Furthermore, for simplicity, follow the text and assume that the y-intercept is zero. Call the X matrix excluding the intercept column `xdot`.

```
xdot = x[.,2:cols(x)];
```

The maximum degree polynomial to be considered is Q = 8 since a polynomial of degree 8 will fit exactly the N + 1 = 9 lag parameters. The polynomial coeffiecients are related to the lag weights by Equation 17.3.1 in the text.
If Q = N the matrix is square and nonsingular, implying that no real restrictions are placed on the lag weights in this case. The lag weights will be restricted to fall on lower and lower polynomial degrees as the higher order polynomial coefficients are set to zero, which also reduces the column dimension of the matrix `hq`. To begin construct the matrix `hq` with Q = N = 8.

```
nvec = seqa(0,1,nhat+1);
qvec = seqa(0,1,nhat+1);
hq = nvec^(qvec');
```

We will sequentially test the hypotheses in Equation 17.3.7 using the test statistic (17.3.8). To implement the test calculate the sum of squared errors $SSE_{N,Q}$ in (17.3.9) and the estimated error variance $\hat{\sigma}^2_{N,Q}$ for $Q = N, N-1, \ldots, 0$, and retain the values in a matrix `store`.

```
store = zeros(nhat+1,3);
test = 0;
do while test le nhat;

q = nhat - test;                /* polynomial degree  */
z = xdot*hq[.,1:q+1];           /* Z as in Eq. 17.3.3 */
ahat = y/z;                     /* OLSE of alpha      */
sse = (y - z*ahat)'(y - z*ahat);/* sse                */
sighat2 = sse/(t-q-1);          /* sighat2            */
store[test+1,.]=q~sse~sighat2;  /* Store values       */
test = test + 1;

endo;
```

Calculate the value of the test statistic in Equation 17.3.8 for each of the tests in (17.3.7). The numerator is the difference between the SSE for the more

restricted model less that of the less restricted model. The denominator is the estimated error variance from the less restricted model. For each F-test one restriction is imposed and is based on $T - (Q + 1)$ degrees of freedom.

```
/*  Eq. 17.3.8       */
lam = (store[2:nhat+1,2] - store[1:nhat,2])
       ./ store[1:nhat,3];

df1 = ones(8,1);                 /*  numerator df     */

i = seqa(1,1,8);
q = nhat - i;
df2 = t - (q + 1);               /*  denominator df   */

format 8,4;                      /*  print results    */
"----i------q------F-stat------p-value----";
i~q~lam~cdffc(lam,df1,df2);
```

Based on these tests we would choose a polynomial of degree Q = 3 for the lag weights. To give an idea of the shapes imposed on the lag weight distributions calculate the unrestricted lag weights, and those for the second and third degree polynomials, and plot them as in Figure 17.3. First, the unrestricted estimates are simply the least squares parameter estimates of Equation 17.2.9 excluding the intercept.

```
b0 = y/xdot;
```

The estimates falling on the third degree polynomial are found by estimating Equation 17.3.3 with 4 columns (Q + 1) retained in Z and then making the transformation in Equation 17.3.4 to obtain the estimates of the lag weights.

```
a3 = y/(xdot*hq[.,1:4]);
b3 = hq[.,1:4]*a3;
```

The estimates for the second degree polynomial are based on (17.3.3) with 3 columns of Z retained and then the transformation to the lag weights.

```
a2 = y/(xdot*hq[.,1:3]);
b2 = hq[.,1:3]*a2;
```

Plot the lag weights against the values $i = 0, \ldots, 10$.

```
library qgraph;
xy(nvec,b0~b2~b3);
```

To obtain estimated standard errors of the estimated lag weights estimate the error variance from the residuals from (17.3.3) corresponding to the third degree polynomial.

```
q = 3;
hq3 = hq[.,1:q+1];
z = xdot*hq3;
sighat2 = (y - z*a3)'(y - z*a3)/(t - q - 1);
```

Obtain the estimated covariance matrix of the polynomial coefficients.

```
cova3 = sighat2*invpd(z'z);
```

Then use Equation 17.3.6 to obtain the covariance matrix of the estimated lag weights. Then construct standard errors, t-statistics and p-values.

```
covb3 = hq3*cova3*hq3';
stderr = sqrt(diag(covb3));
tstat = b3 ./ stderr;
pval = 2*cdftc(tstat,t-q-1);
b3~stderr~tstat~pval;
```

Compare these results to the unrestricted OLS results.

```
sighat2 = (y - xdot*b0)'(y - xdot*b0)/(t - cols(xdot));
covb0 = sighat2 * invpd(xdot'xdot);
stderr = sqrt(diag(covb0));
tstat = b0 ./ stderr;
pval = 2*cdftc(tstat,t-cols(xdot));
b0~stderr~tstat~pval;
```

17.4 Infinite Distributed Lags

In this Section an infinite distributed lag model is described. The statistical model for the geometric lag is given in Equation 17.4.12. To illustrate estimation of this model we will follow the text and generate 5 samples of data using parameter values in Equation 17.4.19. First read 100 random n(0,1) values from the official random numbers contained in the file NRANDOM.DAT to be used as the observations on X and then read an additional 500 values to constitute 5 samples of size 100 on the random disturbance U.

```
open f1 = nrandom.dat;
xvec = readr(f1,100);
u = readr(f1,500);
f1 = close(f1);
u = reshape(u,5,100)';
```

Create a (100 x 5) matrix of zeros that will contain the y-values.

```
y = zeros(100,5);
```

The first observation on y, using (17.4.19), depends on the values of y and u in the time period zero. Assume that these "pre-sample" values are zero and construct the first observation on y, for all 5 samples.

```
y[1,.] = xvec[1,1] + u[1,.];
```

For observations 2,...,100 use Equation 17.4.19 to generate the values of y.

```
obs = 2;
do while obs le 100;
y[obs,.] = xvec[obs,.] + 0.5*y[obs-1,.]
                       + u[obs,.] - 0.5*u[obs-1,.];
obs = obs + 1;
endo;
```

The estimators of the model's parameters will be the least squares estimator, Equation 17.4.13, the instrumental variable estimator (17.4.17) and the maximum likelihood estimator described in Section 17.4.2c. First we will consider the use of OLS and the instrumental variables estimator. In each case only the complete observations will be used. This means that each sample "begins" with observation 2. We will obtain estimates for samples of size 20, 50, and 100 within a DO-LOOP for each sample 1 through 5. Also, for the present assume that the model's intercept is known to be zero.

```
nsam = 1;                          /* begin do-loop      */
do while nsam le 5;

x = xvec~(0|y[1:99,nsam]);         /* full X matrix      */

b20 =  y[2:20,nsam]/x[2:20,.];  /*  OLS on 20 obs     */
b50 =  y[2:50,nsam]/x[2:50,.];  /*  OLS on 50 obs     */
b100 = y[2:100,nsam]/x[2:100,.];/*  OLS on 100 obs    */

z = xvec~(0|xvec[1:99,.]);         /* full instrument    */
                                   /* matrix in 17.4.18  */

z20 = z[2:20,.];                   /*  IV on 20 obs      */
biv20 = inv(z20'x[2:20,.])*z20'*y[2:20,nsam];

z50 = z[2:50,.];                   /*  IV on 50 obs      */
biv50 = inv(z50'x[2:50,.])*z50'*y[2:50,nsam];

z100 = z[2:100,.];                 /*  IV on 100 obs     */
biv100 = inv(z100'x[2:100,.])*z100'*y[2:100,nsam];

"sample" nsam;                     /*  Print results     */
(b20|biv20)';
```

```
(b50|biv50)';
(b100|biv100)';

nsam = nsam + 1;
endo;
```

Compare these results to the OLS and IV estimates in Table 17.4. Before obtaining ML estimates, carry out a Monte Carlo experiment comparing the OLS and IV estimation procedures. We will use the **GAUSS** random number generator for this exercise, so the histogram results we obtain will not be identical to those in Figures 17.4 and 17.5, but they will be similar. Essentially we will simply repeat the steps carried out above for 100 samples instead of 5 and collect all the results. The data generation will be carried out in sets of 50 samples each because of the storage limits of personal computers.

```
open f1 = nrandom.dat;            /*  create X          */
xvec = readr(f1,100);
f1 = close(f1);

y1 = zeros(100,50);               /*  storage matrices  */
y2 = zeros(100,50);

u1 = rndn(100,50);                /*  random numbers    */
u2 = rndn(100,50);

y1[1,.] = xvec[1,1] + u1[1,.];    /* the first obs on y */
y2[1,.] = xvec[1,1] + u2[1,.];

obs = 2;                          /*  obs 2, ..., 100   */
do while obs le 100;

y1[obs,.] = xvec[obs,.] + 0.5*y1[obs-1,.]
            + u1[obs,.] - 0.5*u1[obs-1,.];

y2[obs,.] = xvec[obs,.] + 0.5*y2[obs-1,.]
            + u2[obs,.] - 0.5*u2[obs-1,.];

obs = obs + 1;
endo;

u1 = 0;                       /* clear storage area      */
u2 = 0;

b20    = zeros(2,100);        /* define storage matrices */
b50    = zeros(2,100);
b100   = zeros(2,100);
```

```
biv20  = zeros(2,100);
biv50  = zeros(2,100);
biv100 = zeros(2,100);


iter = 1;                  /*  begin estimation do-loop */
do while iter le 100;

if iter le 50;             /* define y and sample index */
     y = y1; nsam = iter;
else;
     y = y2; nsam = iter - 50;
endif;


x=xvec~(0|y[1:99,nsam]);          /*  X matrix          */
b20[.,iter] =  y[2:20,nsam]/x[2:20,.];   /* OLS, 20 obs */
b50[.,iter] =  y[2:50,nsam]/x[2:50,.];   /* OLS, 50 obs */
b100[.,iter] = y[2:100,nsam]/x[2:100,.]; /* OLS, 100 obs*/



z=xvec~(0|xvec[1:99,.]);          /*  Z matrix          */

z20 = z[2:20,.];                  /*  IV on 20 obs      */
biv20[.,iter] = inv(z20'x[2:20,.])*z20'*y[2:20,nsam];

z50 = z[2:50,.];                  /*  IV on 50 obs      */
biv50[.,iter] = inv(z50'x[2:50,.])*z50'*y[2:50,nsam];

z100 = z[2:100,.];                /*  IV on 100 obs     */
biv100[.,iter] = inv(z100'x[2:100,.])*z100'*y[2:100,nsam];

iter;
iter = iter + 1;
endo;
```

Now calculate the means and standard deviations of the OLS estimates.

```
meanc((b20|b50|b100)')~stdc((b20|b50|b100)');
```

Note that the OLS estimator is biased for the parameter λ.
Define the break points for the histograms in Figure 17.5 for γ (v1) and λ (v2).

```
let v1 = .7 .9 1.1 1.3;
let v2 = 0 .2 .4 .6;
```

Use the "window" option to construct 6 panels and plot the percentage of estimated values falling in each interval.

```
library qgraph;
beggraph;
window(3,2);
{c,m,freq} = histp(b20[1,.]',v1);
{c,m,freq} = histp(b20[2,.]',v2);
{c,m,freq} = histp(b50[1,.]',v1);
{c,m,freq} = histp(b50[2,.]',v2);
{c,m,freq} = histp(b100[1,.]',v1);
{c,m,freq} = histp(b100[2,.]',v2);
endgraph;
```

Repeat the process for the instrumental variable estimator. First calculate the means and standard deviations of the estimates, and note that the IV estimator estimates both parameters well, on average.

```
meanc((biv20|biv50|biv100)')~stdc((biv20|biv50|biv100)');
```

Again obtain the histograms similar to Figure 17.4. The break points for λ are redefined.

```
let v2 = .2 .4 .6 .8;
graphset;
beggraph;
window(3,2);
{c,m,freq} = histp(biv20[1,.]',v1);
{c,m,freq} = histp(biv20[2,.]',v2);
{c,m,freq} = histp(biv50[1,.]',v1);
{c,m,freq} = histp(biv50[2,.]',v2);
{c,m,freq} = histp(biv100[1,.]',v1);
{c,m,freq} = histp(biv100[2,.]',v2);
endgraph;
```

Maximum likelihood estimation of the geometric lag model is simplified by the fact that for a given value of λ the model is linear in the parameters as indicated in Equation 17.2.24 with variables as defined just below Equation 17.2.24. Write a proc that constructs the artificial variables z1 and z2, estimates the model (including an intercept this time) and returns the sum of squared errors and OLS estimates of Model 17.4.25. The arguments of PROC `SSELAM` are y, the vector X and the value of λ. Note that z1 (the second variable in the matrix `zlam`) is calculated recursively following the expression below (17.4.24). Place PROC `SSELAM` in a convenient file and run it.

```
proc (2) = SSELAM(y,x,lam);
     local *;
```

```
     t = rows(x);
     zlam = ones(t,3);
     obs = 1;
     do while obs le t;

          if obs == 1;
               zlam[obs,2] =  x[1,1];
               zlam[obs,3] = lam;
          else;
               zlam[obs,2] = x[obs,1] + zlam[obs-1,2]*lam;
               zlam[obs,3] = lam^obs;
          endif;
     obs = obs + 1;
     endo;

     yt = y[2:t,1];
     zt = zlam[2:t,.];
     blam = yt/zt;
     sse = (yt - zt*blam)'(yt - zt*blam);
     retp(sse,blam);
endp;
```

Place the original data back in memory.

```
          open f1 = nrandom.dat;
          xvec = readr(f1,100);
          u = readr(f1,500);
          f1 = close(f1);
          u = reshape(u,5,100)';

          y = zeros(100,5);

          y[1,.] = xvec[1,1] + u[1,.];

          obs = 2;
          do while obs le 100;
          y[obs,.] = xvec[obs,.] + 0.5*y[obs-1,.]
                                 + u[obs,.] - 0.5*u[obs-1,.];
          obs = obs + 1;
          endo;
```

Now, use PROC `SSELAM` to compute the sum of squared errors for a range of λ values between zero and one for each of the 5 samples constructed earlier and each of the 3 sample sizes. This is fairly long so you will want to place the code in a file and then run it.

```
nsam = 1;                /* loop controlling sample  */
do while nsam le 5;

samsize = 1;             /* loop controlling
                                        sample size */
do while samsize le 3;

if samsize == 1;        /*  select sample size       */
      t = 20;
elseif samsize == 2;
      t = 50;
else;
      t = 100;
endif;


ssevec=zeros(9,3); /* initialize storage matrix for sse
                    and estimates of gamma and lambda */

iter = 1;               /*  begin loop varying lambda */
do while iter le 9;
lam = iter/10;

{sse,blam} = sselam(y[1:t,nsam],xvec[1:t,.],lam);

gam = blam[2,1];              /*  store values        */
ssevec[iter,.]=gam~lam~sse;

iter = iter + 1;
endo;

               /* Find estimates that minimize SSE   */

parm = ssevec[minindc(ssevec[.,3]),.];

t~parm;         /* print ML estimates and sample size */

samsize = samsize + 1;
endo;

nsam = nsam + 1;
?;
endo;
```

Compare these ML results to those in Table 17.4.

Chapter 18

Multiple-Time Series

18.1 Background

In this chapter time series techniques are applied to several economic variables simultaneously. This allows the possibility that each random variable is affected by and related to the rest. The specific model used to describe these multiple time series is a Vector Autoregressive process, or VAR for short.

18.2 Vector Autoregressive Processes

In this Section the basic properties of a VAR(p) process for a vector of M variables are explored. One important property is stationarity. A VAR(p) process is stationary if the roots of polynomial (in z) defined by Equation 18.2.2 are "outside the complex unit circle". That is, if a root of the polynomial is real it must be greater than one in absolute value. If a root is complex, and of the form $r = a + bi$, where $i = \sqrt{-1}$, then $\sqrt{a} + \sqrt{b} > 1$.
In Equation 18.2.4 an example of the problem is given for a VAR(1) process. The matrix theta is given in Equation 18.2.5 and the polynomial is given just below. Define the matrix theta and the polynomial coefficients. Then use the **GAUSS** function POLYROOT to obtain the real and complex roots.

```
let theta1[2,2] = .008 .461
                  .232 .297;
a0 = 1;
a1 = sumc(diag(theta1));
a2 = det(theta1);
a= a2|-a1|a0;
lam = polyroot(a);
lam;
```

In this case the roots are real and greater than 1 in absolute value.

18.3 Estimation and Specification of VAR Processes

In this Section estimators for the parameters of a VAR process are suggested and their properties explored. In addition model selection procedures, for the order of the process, are explained. First LOAD the data in file TABLE18.1 and graph it against time.

```
load dat[75,2] = table18.1;
y1 = dat[.,1];
y2 = dat[.,2];
v = seqa(1,1,75);
```

Use the Quick-graphics command WINDOW to define two panels as in Figure 18.1.

```
library qgraph;
graphset;
beggraph;
window(2,1);
xy(v,y1);
xy(v,y2);
endgraph;
```

The order of the VAR process for the M = 2 time series will be selected using the AIC and SC criteria developed in Section 17.2. In the context of time series they are defined in Equations 18.3.15 and 18.3.16. Following the text we will assume that the maximum order is n = 4. Create vectors containing the current and lagged values of the two variables.

```
y1t = y1[5:71,1];
y1lag1 = y1[4:70,1];
y1lag2 = y1[3:69,1];
y1lag3 = y1[2:68,1];
y1lag4 = y1[1:67,1];

y2t = y2[5:71,1];
y2lag1 = y2[4:70,1];
y2lag2 = y2[3:69,1];
y2lag3 = y2[2:68,1];
y2lag4 = y2[1:67,1];
```

Place the current values of y1 and y2 in a matrix y and define the " X " matrix as in (18.3.2).

```
y = y1t~y2t;
t = rows(y1t);
x = ones(t,1)~y1lag1~y2lag1~y1lag2~y2lag2~y1lag3~y2lag3~y1lag4~y2lag4;
```

Now write a proc that will estimate a VAR process given the matrices y and X. Note that the parameter estimates can be obtain for all equations simultaneously by using GAUSS DIVISION (/) since the X matrix is the same for each of the equations (18.3.4). PROC `VAR` is long so place the code in a file and run it to place in memory.

```
proc (2) = VAR(y,x);
     local *;

     m = cols(y);
     t = rows(y);
     k = cols(x);
     b = y/x;                          /*  Eq. 18.3.4          */
     e = y - x*b;                      /*  obtain residuals    */
     sighat = e'e/(t - k);             /*  Eq. 18.3.11         */
     sigtheta = sighat .*. invpd(x'x); /*  Eq. 18.3.13         */
     stderr = sqrt(diag(sigtheta));

     i = 1;                            /*  begin loop to print */
     do while i le m;
     format 7,3;
     "Eq    " i;
     "est   " b[.,i]';
     "std   " stderr[k*(i-1)+1:i*k,.]';
                                      /*  t-stats for later use */

     tstat = b[.,i] ./ stderr[k*(i-1)+1:i*k,.];
     pval = 2*cdftc(abs(tstat),t-k);
     "tstat " tstat';
     "pval  " pval';
     ?;
     i = i + 1;
     endo;
     format 8,4;
     sighat;
     retp(b,sighat);
endp;
```

Now use PROC `VAR` to estimate the parameters of the VAR(4) process. Compare your results to those in Equation 18.3.14.

```
{parm4,sighat4} = var(y,x);
```

To determine the order of the VAR process write a proc that will calculate the model selection criteria in Equations 18.3.15 and 18.3.16. In addition calculate the determinant of the contemporaneous covariance matrix. The arguments of PROC `VARSTAT` are the matrix y and the X matrix which contains a column of ones and lagged values of y for orders n = 0, 1, 2, 3 and 4.

```
proc  VARSTAT(y,x);
     local *;

     m = cols(y);
     t = rows(y);
     n = (cols(x) - 1)/2;
     b = y/x;                                  /*  Estimate parms */
     e = y - x*b;                              /*  residuals      */
     sig = e'e/t;                              /*  Eq. 18.3.17    */
     aic = ln(det(sig)) + 2*(m^2)*n/t;         /*  Eq. 18.3.15    */
     sc = ln(det(sig)) + (m^2)*n*ln(t)/t;      /*  Eq. 18.3.16    */
     format 8,4;
     "----n-------det--------aic--------sc";?;
     n~det(sig)~aic~sc;
     ?;
     "     sighat    ";
     sig;

     retp("");
endp;
```

Run PROC `VARSTAT` to place it in memory and then calculate the selection criteria for successively larger orders of lags. Compare your results to those in Table 18.2. Note that these tests are all based on the data created above which has T = 67 complete observations.

```
x = ones(t,1);                             /*    n = 0   */
varstat(y,x);

x = x~y1lag1~y2lag1;                       /*    n = 1   */
varstat(y,x);

x = x~y1lag2~y2lag2;                       /*    n = 2   */
varstat(y,x);

x = x~y1lag3~y2lag3;                       /*    n = 3   */
varstat(y,x);

x = x~y1lag4~y2lag4;                       /*    n = 4   */
varstat(y,x);
```

Since the AIC and SC criteria obtain their minima at lag n = 1 we will use a VAR(1) process to describe the data. Re-estimate the model assuming this lag. Correct the definitions of y and X using T = 70 complete observations. Compare your results to Equation 18.3.18.

```
y1t = y1[2:71,1];
```

```
y1lag1 = y1[1:70,1];

y2t = y2[2:71,1];
y2lag1 = y2[1:70,1];

y = y1t~y2t;
t = rows(y1t);
x = ones(t,1)~y1lag1~y2lag1;

{parm1,sighat1} = var(y,x);
```

18.4 Forecasting Vector Autoregressive Processes

Optimal forecasting in the context of a VAR process means that the forecast mean square error is minimized. The optimal h-step-forecasts are given in Equation 18.4.1 and recursions defined just below that equation. To implement these forecasting techniques separate the estimated parameters PARM1 from the previous section into the intercept (nu) and lag weights (theta1).

```
nu = parm1[1,.]';
theta1 = parm1[2:3,.]';
```

Using starting period T = 71 forecast one and two periods into the future, following Equation 18.4.2.

```
y71 = dat[71,.]';

yhat1 = nu + theta1*y71;
yhat1;

yhat2 = nu + theta1*yhat1;
yhat2;
```

The mean square error matrix of the forecasts is defined in Equations 18.4.3-18.4.5. For the VAR(1) process we are using the MSE matrices are given in Equations 14.4.6.

```
sig1 = sighat1;
sig1;

sig2 = sighat1 + theta1*sighat1*theta1';
sig2;
```

These MSE matrices and the asymptotic normality of the parameter estimators can be used to construct forecast intervals as in (18.4.9) and (18.4.10).

```
lb1 = yhat1 - 1.96*sqrt(diag(sig1));
ub1 = yhat1 + 1.96*sqrt(diag(sig1));
lb1~ub1;

lb2 = yhat2 - 1.96*sqrt(diag(sig2));
ub2 = yhat2 + 1.96*sqrt(diag(sig2));
lb2~ub2;
```

One way to verify the model is to examine its forecasting accuracy. Note that the actual observations for T = 72 and T = 73 fall inside the confidence intervals.

```
dat[72,.]';

dat[73,.]';
```

18.5 Granger Causality

To test for the existence of causality in VAR models, as defined by Granger, standard test procedures can be used. First test the hypothesis that y2 does not cause y1 using the test statistic in Equation 18.5.5. In this context the "unrestricted" estimates are given by the VAR(1) model above. The "restricted" estimates are obtained by estimating the model assuming that the lagged value of y2 does not significantly affect y1. That is, only the lagged value of y1 is present in the model.

```
y1t = y1[2:71,1];          /* define y1 and y1 lagged */
y1lag1 = y1[1:70,1];

t = rows(y1t);             /* define X                */
x = ones(t,1)~y1lag1;

{parmr,sigr} = var(y1t,x);/* restricted estimates    */
```

Compare your restricted estimates to those in Equation 18.5.6. Then calculate the value of the test statistic which is defined just below (18.5.6). Under the null hypothesis that y2 does not cause y1 it has an F-distribution with 1 and T-3 degrees of freedom (asymptotically).

```
lam = (sigr*(t-2) - sighat1[1,1]*(t-3))/sighat1[1,1];
pval = cdffc(lam,1,t-3);
lam~pval;
```

On the basis of this test we reject the null hypothesis at the 1significance. As is noted in the text, for the VAR(1) model an equivalent test can be carried out using the t-statistics obtained when the VAR(1) model was estimated. You may wish to check this if you have not printed out your results.

To illustrate a test of the hypothesis that income does not cause consumption the VAR(4) process initially estimated will be used. The unrestricted estimates are given by (18.3.14) which we estimated earlier. The restricted model is obtained by estimating the model where y1 is determined only by its lagged values. Estimate the model in (18.5.7) and carry out the joint test of significance.

```
y1t = y1[5:71,1];              /* define y and lags   */
y1lag1 = y1[4:70,1];
y1lag2 = y1[3:69,1];
y1lag3 = y1[2:68,1];
y1lag4 = y1[1:67,1];

t = rows(y1t);                 /* define X            */
x = ones(t,1)~y1lag1~y1lag2~y1lag3~y1lag4;

{parmr,sigr} = var(y1t,x);     /* estimate Eq. 18.5.7 */
```

Now carry out the test and note that the hypothesis is rejected at the 1% level.

```
lam = (sigr*(t-5) - sighat4[1,1]*(t-9))/(4*sighat4[1,1]);
pval = cdffc(lam,4,t-9);
lam~pval;
```

18.6 Innovation Accounting and Forecast Error Variance Decomposition

To trace the effect of a shock (or innovation) to the Multiple Time series system a multiplier anaysis is useful. To trace the effect of a shock to income (y2) assume that the values of y1 and y2 in period 0 are (0,1). Recursively predict the time path of the variables as is done following (18.6.1). Note that for simplicity the mean vector (nu) has been dropped.

```
let y0 = 0 1;
y1hat = theta1*y0;
y1hat;

y2hat = theta1*y1hat;
y2hat;
```

If a shock of one standard deviation in income were traced

```
y0 = 0|sqrt(sighat1[2,2]);
y1hat = theta1*y0;
y1hat;

y2hat = theta1*y1hat;
y2hat;
```

To carry out the innovation analysis in the system transformed to have contemporaneously uncorrelated errors we must diagonalize the covariance matrix `sighat1`. As noted in the text there are many ways to do this. First let P be given by the Cholesky decomposition of the inverse of the contemporaneous covariance matrix using the GAUSS command CHOL.

```
p1 = chol(invpd(sighat1));
p1;
```

Check to see if `p1` diagonalizes `sighat1` to an identity matrix, except for rounding error.

```
check = p1*sighat1*p1';
check;
```

Calculate the inverse of P^{-1} and use it to construct the multiplier matrices in (18.6.6). For the VAR(1) process the M matrices are powers of the parameter matrix Θ_1.

```
invp1 = inv(p1);
invp1;

psi0 = eye(2)*invp1;

m1 = theta1;
m2 = theta1*theta1;

psi1 = m1*invp1;
psi1;

psi2 = m2*invp1;
psi2;
```

The income innovation (0,1) in this model is different from the effects in (18.6.3).

```
let w0 = 0 1;
y0 = psi0*w0;
y1 = psi1*w0;
y2 = psi2*w0;
y0~y1~y2;
```

One problem with this analysis is that there are many P matrices that will diagonalize the covariance matrix. For example, taking the inverse of the transposed Cholesky decomposition also "works". Check this.

```
p = chol(sighat1);
p = inv(p');
p;
```

```
check = p*sighat1*p';
check;
```

Naturally the innovation analysis is quite different and the form of P must be selected on the basis of a priori information.
Another innovation analysis determines the percent of the MSE contributed by the innovations. Using Equation 18.6.8 the two-step forecast variance of consumption is

```
f = psi0[1,1]^2 + psi1[1,1]^2 + psi0[1,2]^2 + psi1[1,2]^2;
f;
```

Decomposing the contributions of y1 and y2

```
f1 = psi0[1,1]^2 + psi1[1,1]^2;
f2 = psi0[1,2]^2 + psi1[1,2]^2;
```

The percent contributions are

```
pctown = f1/f;
pctown;

pctinc = f2/f;
pctinc;
```

Chapter 19

Qualitative and Limited Dependent Variable Models

19.1 Introduction

In this Chapter some very useful models are considered. Probit and Logit are designed for situations when the dependent variable takes only two values, usually 1 and 0, indicating that some event did, or did not, occur. Tobit is appropriate when the dependent variable is truncated. That is, for example, it may only be observable if it is positive.

19.2 Binary Choice Models

An example of Maximum Likelihood estimation of the probit model is given on pages 793-794. The data for the example is generated as follows. First, LOAD the design matrix in Equation 5.10.2. It is in the file JUDGE.X. Then stack it four times to produce a design matrix with T = 80 observations.

```
load x[20,3] = judge.x;
x = x|x|x|x;
t = rows(x);
k = cols(x);
```

Create an error vector of N(0,1) disturbances using the "official" normal random disturbances in NRANDOM.DAT.

```
open f1 = nrandom.dat;
e = readr(f1,80);
f1 = close(f1);
```

Let beta take the indicated values (0,3,-3) and construct the variable ystar, which is unobservable to the economic researcher.

```
let beta = 0 3 -3;
ystar = x*beta + e;
```

What is observable is the binary variable y_t, which takes the value of 1 or 0 depending on whether `ystar` is positive or not. Compare the 80 values produced to the values in Table 19.4 for y_1, y_2, y_3 and y_4.

```
y = ystar .> 0;
format 4,2;
y';
```

To estimate the probit model using the Newton-Raphson algorithm, (19.2.20), first write PROC `PROBITLI` which returns the value of the log-likelihood function. It assumes that X and y are in memory and takes as argument an estimate of beta. It uses the **GAUSS** function CDFN which returns the value of the CDF for a N(0,1) random variable.

```
proc PROBITLI(b);               /*  See Eq. 19.2.18
*/ local cdf,li;
     cdf = cdfn(x*b);
     li = y .* ln(cdf) + (1-y) .* ln(1-cdf);
     retp(sumc(li));
endp;
```

At this point there are several ways to proceed. The general optimization algorithms of Chapter 12 could be used to maximize the log-likelihood function or an algorithm specifically for probit could be written, which entails programming the first and second derivatives. We will follow both paths. First, PROC `PROBIT` is written which returns the probit ML estimates and their asymptotic covariance matrix using (19.2.20) and the derivatives given in (19.2.19) and (19.2.21). Its arguments are X, y and PROC `PROBITLI`. Place the PROC in a convenient file and run it.

```
proc (2) = PROBIT(x,y,&PROBITLI);
local probitli:proc;
local t,k,b,iter,crit,pdf,cdf,g,d,h,db,bn,s,ofn1,ofn2,covb,std;

iter = 1;                          /*   define constants         */
crit = 1;
b = y/x;                           /*  initial param. estimates  */

do until (iter ge 50) or (crit le 1e-6);

pdf = pdfn(x*b);                        /*  f and F  */
cdf = cdfn(x*b);

                /*  Gradient vector. See Eq. 19.2.19    */
```

```
g = y.*(pdf./cdf).*x - (1-y).*(pdf./(1-cdf)).* x;
g = sumc(g);

                /*  Hessian Matrix.  See Eq. 19.2.21    */
                /*  H = -X'DX where D is diagonal       */

d = pdf.*( (y.*(pdf+(x*b).*cdf)./cdf^2)
          +((1-y).*(pdf-(x*b).*(1-cdf))./(1-cdf)^2));
H = -(x .* d)'x;

db = -inv(H)*g;                 /*  Full Newton-Raphson step  */
gosub step;                     /*  Determine step length     */
bn = b + s*db;                  /*  New estimates             */
crit = maxc(abs(db));           /*  Convergence criterion     */
gosub prnt;                     /*  Print iteration results   */
b = bn;                         /*  Replace old with new      */
iter = iter + 1;                /*  Increment iteration       */
endo;                           /*  End do-loop               */

?;                              /*  Print final results       */
"Final results:  " ;
"Estimates:      " b';
covb = -inv(H);                 /*  Define Covariance matrix  */
std = sqrt(diag(covb));         /*  Asymptotic Std. Errors    */
"Std. Errors:    " std';
"Asy. t-values:  " (b./std)';   /*  Asymptotic t-values       */

retp(b,covb);

step:                           /*  Determine step length     */
s = 2;
ofn1 = 0;
ofn2 = 1;
do until ofn1 >= ofn2;
s = s/2;
ofn1 = probitli(b+s*db);
ofn2 = probitli(b+s*db/2);
endo;
return;

prnt:                           /*  Print iteration results   */
format 4,2; "iter = " iter;;
format 3,2; "step length =" s;;
format 10,6; "Likelihood =" probitli(bn);
format 10,6; " b = " bn';?;
```

```
return;

endp;                                   /*  End    */
```

Use PROC PROBIT to estimate the parameters of our model

```
{bp,covbp} = PROBIT(x,y,&PROBITLI);
```

Instead of writing a complete new PROC, we could have used PROC MAXM from Chapter 12. Make sure PROC MAXM is in memory and obtain the maximum likelihood estimates of the parameters using this Newton-Raphson algorithm based on numerical derivatives. Use the OLS estimates as starting values.

```
b = y/x;
bp = MAXM(b,&probitli);
```

You should observe that using numerical derivatives is slower. In this case the numerical derivatives provide a good approximation to the Hessian when evaluated at the final estimates. That is because the log-likelihood function of the probit model is strictly concave.
Given this experience estimation of the logit model is easy. Write PROC LOGITLI that returns the value of the log-likelihood function for the logit model.

```
proc  LOGITLI(b);                   /*  See  Eq. 19.2.18 */
local cdf,li;
     cdf = 1 ./ (1 + exp(-x*b));
     li = y .* ln(cdf) + (1-y) .* ln(1-cdf);
retp(sumc(li));
endp;
```

Use PROC MAXM to obtain the ML estimates of the parameters of the logit model.

```
b = y/x;
bl = MAXM(b,&LOGITLI);
```

Once again there may be a question how close the approximate Hessian is to the true one. The Hessian for the logit model is given in (19.2.22).

```
pdf = exp(-x*bl) ./ (1+exp(-x*bl))^2;
H = -(x .* pdf)'x;
std = sqrt(diag(-inv(H)));
std';
```

In this case the approximate Hessian is very close to the true one at the final estimates.
The likelihood-ratio test for the "overall" significance of the model is described on the top of page 794. First calculate the value of the log- likelihood function at the ML estimates for the probit model.

```
l1 = probitli(bp);
```

Under the hypothesis the X matrix is a column of ones. Use the sample mean of y as the initial estimate of the remaining parameter, β_1, and obtain the ML estimate for this restricted model.

```
x = ones(t,1);
b = meanc(y);
br = MAXM(b,&probitli);
```

Calculate the value of the log-likelihood for the restricted model and then the value of the test statistic.

```
l2 = probitli(br);
lr = 2*(l1-l2);
pval = cdfchic(lr,t-k);
lr~pval;
```

Calculate the value of the "Pseudo"-R^2.

```
r2 = 1-(l1/l2);
r2;
```

Calculate the values of the partial derivatives at `xstar`.

```
let xstar = 1 .69 .69;
partials = pdfn(xstar'bp) .* bp;
partials;
```

19.3 Models with Limited Dependent Variables

In this Section the Tobit model is presented. Construct the data in Table 19.2 as follows. First construct X and a vector of 20 N(0,16) random errors.

```
x = ones(20,1)~seqa(1,1,20);
t = rows(x);
k = cols(x);
open f1 = nrandom.dat;
e = readr(f1,20);
f1 = close(f1);
e = 4*e;
```

Using the given parameter values construct ystar and y.

```
let beta = -9 1;
ystar = x*beta + e;
z = (ystar .> 0);
y = z .* ystar;
y~x;
```

Obtain the OLS estimates using all the data and ignoring the truncated nature of the dependent variable. Compare to column 1 of Table 19.3.

```
b = y/x;
sig2 = (y - x*b)'(y - x*b)/(t - k);
std = sqrt(diag(sig2*invpd(x'x)));
b~std~(b./std);
```

Delete the observations that have y = 0 and use OLS again. Compare the results to the second column of Table 19.3.

```
dat = y~x;

/*  Sort the data and place the complete observations first,
         then delete the rest   */

dat = rev(sortc(dat,1));
t1 = sumc(z);
yt = dat[1:t1,1];
xt = dat[1:t1,2:cols(dat)];

/*  Apply OLS  */

bt = yt/xt;
sigt = (yt - xt*bt)'(yt - xt*bt)/(t1 - k);
stdt = sqrt(diag(sigt*invpd(xt'xt)));
bt~stdt~(bt./stdt);
```

To carry out ML estimation we will use `MAXM`. First write PROC `TOBITLI` which returns the value of the Tobit log-likelihood function given in Equation 19.3.6. Its argument is an initial set of estimates for β and σ^2, in that order.

```
proc TOBITLI(p0);
local k1,b,sig,z,t1,l1,l2,li;

k1 = rows(p0);               /*  sort out estimates  */
b = p0[1:k1-1,1];
sig = sqrt(p0[k1,1]);

z = (y .> 0);              /* number of complete obs */
t1 = sumc(z);

l1 = ln(1-cdfn(x*b/sig)) .* (1-z);    /* first term  */
l2 = (((y-x*b)^2)./(2*sig^2) ) .* z;  /* last term   */

li = sumc(l1)-.5*t1*(ln(2*pi)+ln(sig^2))-sumc(l2);
retp(li);
endp;
```

Using as initial estimates the results of OLS applied to the complete sample, obtain the ML estimates of the parameters.

```
p0 = b|sig2;
param = MAXM(p0,&TOBITLI);
```

The asymptotic covariance matrix for the ML estimates is given by the inverse of the information matrix in Equation 19.3.8. Write PROC TOBITI to compute it. Its argument is the set of ML estimates.

```
proc TOBITI(p0);
local k1,b,sig,z,pdf,cdf,a,b,c;

k1 = rows(p0);          /*  sort out estimates   */
b = p0[1:k1-1,1];
sig = sqrt(p0[k1,1]);

z = (x*b)./sig;          /*  argument of std. normal */

pdf = pdfn(z);           /*    f    */
cdf = cdfn(z);           /*    F    */

/*  write a, b, c as column vectors   */

a = -(z.*pdf - pdf^2 ./(1-cdf) -cdf) ./(sig^2);

b = ((z^2 .* pdf)+pdf-(z .* pdf^2)./(1-cdf))./(2*sig^3);

c = -((z^3 .* pdf)+z.*pdf-((z^2) .* pdf^2)
    ./(1-cdf)-2*cdf)./(4*sig^4);

/*  construct the components of (19.3.8)  */

a = (x.*a)'x;
b = sumc(x.*b);
c = sumc(c);
retp( (a~b)|(b'~c) );

endp;
```

Use PROC TOBITI to calculate the estimate of the asymptotic covariance matrix, standard errors and t-values.

```
covp = inv(TOBITI(param));
covp;
std = sqrt(diag(covp));
param~std~(param./std);
```

In Equations (19.3.9) and (19.3.10) various regression functions and their partial derivatives are shown.

```
b = param[1:k,1];
sig = sqrt(param[k+1,1]);
xbar = meanc(x);

z = (xbar'*b)./sig;

cdfn(z);                              /*  19.3.10b  */

m = 1 - z*pdfn(z)/cdfn(z)             /*  19.3.10c  */
    -(pdfn(z)/cdfn(z))^2;
m;
```

Chapter 20

Biased Estimation

20.1 Statistical Decision Theory

In this Section basic concepts of decision theory are presented.

20.2 Combining Sample and Nonsample Information

Various ways of incorporating sample and nonsample information are discussed in this Section. The first example is given in Section 20.2.1b. It uses the sampling model described in Section 6.1.5. LOAD the (20 x 3) design matrix in JUDGE.X, specify the true parameter vector, β, and construct a sample of y values using N(0,.0625) random disturbances. Examine the data.

```
load x[20,3] = judge.x;
t = rows(x);
k = cols(x);
let beta = 10 0.4 0.6;
open f1 = nrandom.dat;
e = readr(f1,20);
f1 = close(f1);
e = sqrt(0.0625)*e;
y = x*beta + e;
format 10,6;
y~x;
```

Construct the linear restrictions in (20.2.1a) and obtain the RLS estimates using (20.2.5).

```
let r[1,3] = 0 1 1;                    /*   20.2.1a   */
rr = 1;
```

```
b = y/x;
sinv = invpd(x'x);
q = invpd(r*sinv*r');
br = b + sinv*r'*q*(rr - r*b);        /*   20.2.5    */
```

The covariance matrix for the RLS estimator is given in Equation 20.2.7. To estimate the error variance either the usual, unbiased estimator can be used or the sum of squared errors can be based on the RLS estimates and the degrees of freedom corrected for the fact that $J = rank(R)$ fewer parameters need be estimated.

```
sighat2 = (y - x*b)'(y - x*b)/(t - k);

j = rows(r);                          /*  based on RLS */
sig2 = (y - x*br)'(y - x*br)/(t - k + j);
```

This provides two estimates of the RLS covariance matrix.

```
/*  Eq. 20.2.7    */
covbr = sighat2 * (sinv - sinv*r'*q*r*sinv);
covbr2 = sig2 * (sinv - sinv*r'*q*r*sinv);
```

Compare the OLS and RLS estimates.

```
format 8,5;
b; sighat2*sinv;
br; covbr; covbr2;
```

For later use, write PROC `RLS` to produce the RLS estimates and covariance matrix given `y`, `x`, `r` and `rr`. Use the restricted residuals to estimate the error variance.

```
proc (2) = RLS(x,y,r,rr);
local t,b,sinv,q,br,sser,sig2,covbr,stderr,sseu,
                                    sighat2,fstat;

     t = rows(x); k = cols(x); j = rows(r);
     b = y/x;
     sinv = invpd(x'x);
     q = invpd(r*sinv*r');
     br = b + sinv*r'*q*(rr - r*b);
     sser = (y - x*br)'(y - x*br);
     sig2 = sser/(t-k+j);
     covbr = sig2 * (sinv - sinv*r'*q*r*sinv);
     stderr = sqrt(diag(covbr));
```

```
    format 10,4;

    "RLS results";
    ?;
    "R || rr";
    r~rr;
    ?;
    "Est           :" br';
    "Std. Err.     :" stderr';

    sseu = (y - x*b)'(y - x*b);
    sighat2 = sseu ./ (t - k);
    fstat = (sser - sseu)/(j * sighat2);
    ?;
    " F-statistic " fstat;
    " pval=       " cdffc(fstat,j,t-k);
    retp(br,covbr);
endp;
```

The use of stochastic contraints is illustrated on page 819. It uses the same sampling model. Write the restrictions (20.2.13a) and the precision matrix Ω.

```
let r[2,3] = 0 1 0
             0 0 1;
let rr = .5 .5;
om = eye(2) ./ 64;
```

Obtain parameter estimates using (20.2.15) and the covariance matrix in (20.2.17). This example assumes $\sigma^2 = .0625$ is known.

```
q = invpd(x'x + r'*invpd(om)*r);
btilde = q*(x'y + r'*invpd(om)*rr);
covbtil = (.0625) * q;
btilde; covbtil;
```

An example using inequality constraints is given on p.822. It is simply noted that the OLS estimates violate the single restriction in (20.2.24a) and that the inequality RLS estimates are equal to the RLS estimates.

```
b';
br';
```

In Section 20.2.3d Bayesian analysis with inequality restrictions is considered. Given the sampling model in Equation 20.2.37 and a noninformative prior is assumed then the posterior is given by (20.2.39). Write a PROC that returns the value of the posterior given the least squares estimate `b`, the sum of the squared x values `xsum` and the multiplicative constant `c`.

```
c = 1;
b = .95;
xsum = 196;

proc POST(beta);
     local g;
     g  =  c.*sqrt(xsum)
           .*exp(-0.5*xsum*(beta-b)^2)./sqrt(2*pi);
     retp(g);
endp;
```

In order to calculate the normalizing constant c for the truncated normal posterior in (20.2.41), let $c = 1$ with the values for `b` and the sum of squares at the bottom of page 827. Use the **GAUSS** function INTQUAD1 to numerically integrate the posterior density from 0 to 1. The normalizing constant is the reciprocal of this value as given on page 828.

```
_intord = 20;
xl = 1|0;
y = intquad1(&POST,xl);
y;
c = 1/y;
c;
```

Plot the normal and truncated normal posterior functions.

```
beta = seqa(.7,.005,101);
n1 = POST(beta);         /*  normal posterior        */
n2 = n1*c .* (beta .< 1);/*  truncated normal post'r */

library qgraph;
xy(beta,n1~n2);
```

To compute the Bayesian point estimate of beta the mean of the posterior distribution must be obtained. The weighted posterior can be numerically integrated over the range [0,1] to compute the expected value of β. PROC `EXPT` returns the value of the weighted density function which is then integrated.

```
proc EXPT(beta);
   local g;
   g  = c.*beta.*sqrt(xsum)
        .*exp(-0.5*xsum*(beta-b)^2)./sqrt(2*pi);
   retp(g);
endp;

betabar = intquad1(&expt,xl);
format 10,6;
betabar;
```

To compute the Bayesian 95% highest posterior density interval for the truncated posterior calculate the probability mass over a rough grid of values and then refine the search.

```
lb = seqa(.7,.01,20);
ub = ones(20,1);
xl = ub'|lb';
prob = intquad1(&POST,xl);
lb~prob;
```

The lower bound of the 95% interval is near .82. Examine this area more carefully.

```
lb = seqa(.81,.001,20);
xl = ub'|lb';
prob = intquad1(&POST,xl);
lb~prob;
```

The example is then repeated for a different value of `b`, which violates the prior information.
Find the normalizing constant.

```
b = 1.05;
c = 1;
xsum = 196;

xl = 1|0;
y = intquad1(&POST,xl);
y;
c = 1/y;
c;
```

Graph the posteriors.

```
beta = seqa(.8,.005,101);
n1 = POST(beta);
n2 = n1*c .* (beta .< 1);
xy(beta,n1~n2);
```

Find the mean of the truncated posterior.

```
betabar = intquad1(&expt,xl);
betabar;
```

Find the 95% HPD interval.

```
lb = seqa(.8,.01,20);
ub = ones(20,1);
xl = ub'|lb';
y = intquad1(&POST,xl);
lb~y;
```

Search near .89 for the lower bound of the interval.

```
lb = seqa(.88,.001,20);
xl = ub'|lb';
y = intquad1(&POST,xl);
lb~y;
```

Finally, let us extend the results to allow for an unknown error variance, as on page 830 of ITPE2. The example used is that in Section 6.1.5. LOAD in the (20 x 3) design matrix found in the file JUDGE.X and examine it.

```
t = 20;
k = 3;
load x[t,k] = judge.x;
x;
```

Now create the values of the dependent variable using the parameter values given in the text and the first 20 "official" normal random numbers, which are transformed to have variance .0625.

```
let beta = 10 .4 .6;
open f1 = nrandom.dat;
e = readr(f1,20);
f1 = close(f1);
e = sqrt(.0625) * e;
y = x*beta + e;
```

Calculate the ML estimates of beta and the unbiased estimator of the error variance.

```
b = y/x;
sighat2 = (y - x*b)'(y - x*b)/(t - k);
covb = sighat2 * invpd(x'x);
```

The idea now is to do the following. If a uniform inequality prior is placed on the parameters, namely that the sum of the second and third parameters is less than or equal to one, the posterior distribution is a multivariate-t which is truncated itself. The parameter estimates are the mean of this truncated joint distribution. Since it is difficult to integrate this function in a straightforward manner, we use a technique called "Monte Carlo Integration". What it amounts to is generating many (we will use 20,000) (K x 1) vectors of βs from the untruncated multivariate-t distribution and then simply keep track of the mean and variance of those beta vectors which satisfy the inequality constraint. Those means and variances are our Bayesian parameter estimates.
The first problem we must address is how to generate random values from a multivariate-t distribution. The method is described in "Further experience in

Bayesian Analysis Using Monte Carlo Integration," by H.K. van Dijk and T. Kloek, *Journal of Econometrics*, vol. 14, pages 307-328, 1980. The procedure is to generate multivariate normal random values from a distribution with mean B and covariance matrix COVB, which we have calculated above, and divide by the square root of a random value from a chi-square distribution which has been divided by its degrees of freedom, (T - K).
To generate the desired multivariate normal distribtion we use a transformation matrix such that P*P' = COVB. That is provided by the transpose of the Cholesky decomposition. Find and check this matrix.

```
pt = chol(covb);
p = pt';
check = p*p';
check;
covb;
```

As noted above we wish to generate 20,000 (K x 1) vectors. This we will do in 25 loops that generates 800 such vectors at a time. We will also use the concept of *antithetic* random numbers which says to use both the positive and negative values of a random number to reduce numerical inaccuracy. Thus the number of random samples in each iteration will be NSAM = 400.

```
niter = 25;
nsam = 400;
totsam = 2*niter*nsam;
```

We will use separate seed values for the multivariate normal and chi-square random deviates to ensure independence of the numerator and denominator.

```
seed1 = 93578421;
seed2 = 17411329;
```

Finally, define storage matrices in which we will accumulate the sum, sum of squares and posterior probability of the inequality constrained parameters.

```
bbar2 = zeros(k,1);
bbaravg = zeros(k,1);
totn = 0;
```

Now we are ready to start. The following code should be placed in a file and executed. It will take a couple of minutes to complete the execution. Also, the results you obtain will not be exactly the same as in the text as a different set of random numbers is used, but they will be close.

```
iter = 1;                          /*  begin loop              */
do while iter le niter;

w = rndns(t-k,nsam,seed2);         /*  create chi.sq.(t-k)     */
```

```
w = sumc(w .* w)';

inc = p*rndns(k,nsam,seed1);       /*  normal(b,covb)          */
inc = inc ./ sqrt( w ./ (t-k));    /*  divide by chi.sq.       */
bbar = (b + inc)~(b - inc);        /*  multi-t                 */

bsum = bbar[2,.] + bbar[3,.];      /*  sum b2 and b3           */
success = (bsum .le 1);            /*  check constraint        */
totn = sumc(success') + totn;      /*  count successes         */
bsat = bbar .* success;            /*  select successes        */
bbaravg = bbaravg + sumc(bsat');   /*  collect sum             */
/*  collect sum of sq.      */
bbar2 = bbar2 + sumc( (bsat.*bsat)' );
?;
"iter     =" iter;                 /*  print iteration         */
"totn     =" totn;                 /*  cumulative total        */
iter = iter + 1;
endo;

postprob = totn/totsam;                          /*  post. prob  */
bbarbar = (bbaravg ./ totn);                     /*  means       */
bbarvar = ((bbar2 - (totn .* bbarbar^2))/totn);/*  variances   */

?;
"totn        =" totn;                            /*  print       */
"postprob    =" postprob;
"bbarbar     =" bbarbar';
"bbarvar     =" bbarvar';
```

20.3 Pretest and Stein Rule Estimators

In this section the sampling properties of the pretest estimator and the Stein-rule estimator are presented.

20.4 Model Specification

In this Section various rules are discussed which have been proposed to aid in the selection of regressors to include in a statistical model. In order to examine the ability of these rules to detect the correct set of regressors a small monte carlo experiment is carried out in Section 20.4.3f. The true model is the one presented in Section 6.1.5 and which has been used earlier in this chapter. Construct 100 samples of y with T = 20 observations using this model. Recall that 250 samples of random errors from a N(0,1) are stored in e1nor.fmt.

```
t = 20;
```

```
load x[20,3] = judge.x;
load xa[20,3] = table20.52;
let beta = 10 0.4 0.6;
load e = e1nor.fmt;
e = sqrt(.0625)*e[.,1:100];
y = x*beta +  e;
```

The matrix y is (20 x 100) with each column representing a sample of size 20. For the purposes of exploration we will add 3 regressors to the X matrix. These are shown in Equation (20.5.2) as x_4, x_5 and x_6 and are contained in the file TABLE20.52 on your data disk.

```
format 10,6;
x = x~xa;
x;                                  /*  Eq. 20.5.2       */
```

We wish to examine the ability of the four criteria based on $\bar{R}^2$, AIC, PC and SC to detect the correct model from the seven sets of regressors listed in Table 20.3. Write a program that will calculate each of the selection criteria for each of the 7 models for all 100 samples.

```
rbar2 = zeros(100,7);           /*  storage matrices  */
aic = zeros(100,7);
sc = zeros(100,7);
pc = zeros(100,7);

x1 = x[.,1:3];              /* 7 model specifications */
x2 = x[.,1:4];
x3 = x[.,1 2 3 5];
x4 = x[.,1 2 3 6];
x5 = x[.,1:5];
x6 = x[.,1 2 3 4 6];
x7 = x[.,1 2 3 5 6];

/*  SST   */
sst = sumc( (y-meanc(y)') .* (y-meanc(y)') );


model = 1;                          /*  start do-loop  */
do while model le 7;
model;
if model == 1; x = x1;              /*  select model  */
elseif model == 2; x = x2;
elseif model == 3; x = x3;
elseif model == 4; x = x4;
elseif model == 5; x = x5;
elseif model == 6; x = x6;
```

```
else; x = x7;
endif;

b = y/x;                              /*  ols estimates  */
k1 = cols(x);                         /*      K1         */
ssei = sumc( (y-x*b) .* (y-x*b) );/*      SSE        */

 /*   Selection criteria as defined in Table 20.2    */

rbar2[.,model] =  1 - ((t-1)/(t-k1)) .* (ssei ./ sst);
aic[.,model] = ln(ssei/t) + 2*k1/t;
sc[.,model] = ln(ssei/t) + k1*ln(t)/t;
pc[.,model] = (ssei/(t-k1)) .* (1+k1/t);

model = model + 1;
endo;                                   /*  end do-loop  */
```

To count the number of times each model is selected under each criterion use the MAXINDC and MININDC to return the indices of respective model choices and count the number taking values 1,...,7.

```
let v = 1 2 3 4 5 6 7;
rbar2F = counts(maxindc(rbar2'),v);
aicF = counts(minindc(aic'),v);
scF = counts(minindc(sc'),v);
pcF = counts(minindc(pc'),v);
table = rbar2f~aicF~pcF~scF;

format 4,2;
table;
```

Chapter 21

Multicollinearity

21.1 Introduction

Economists and other social scientists use nonexperimental data. That is, the data does not come from a controlled experiment. One frequent problem with such data is that it is multicollinear. In this chapter the nature of the multicollinearity problem is discussed. Methods of detection are presented and various "cures" proposed.
An example of the consequences of multicollinearity for the least squares estimator is provided by the Klein-Goldberger consumption function. LOAD the data in file TABLE21.1 on your data disk and check it.

```
load dat[20,5] = table21.1;
format 10,6;
dat;
```

The dependent variable in the model is consumption (C) and the explanatory variables are three types of income (W, P, A). Construct the y vector and X matrix.

```
t = rows(dat);
y = dat[.,2];
x = ones(t,1)~dat[.,3 4 5];
k = cols(x);
```

Use PROC `OLS`, written in Chapter 6, to obtain estimates of the parameters. Make sure that the proc is in memory or where **GAUSS** can find it. Compare your results to those in Table 21.2.

```
{b,covb} = myols(x,y);
```

Construct the 95

```
stderr = sqrt(diag(covb));
lb = b - 2.12 .* stderr;
ub = b + 2.12 .* stderr;
lb~ub;
```

21.2 The Statistical Consequences of Multicollinearity

In this Section the statistical consequences of multicollinearity on the least squares estimator are presented. In particular, multicollinearity adversely affects the sampling variance of the OLS estimator.

21.3 Detecting the Presence, Severity, and Form of Multicollinearity

Various strategies for detecting the nature of multicollinear data are discussed in this Section. In Section 21.3.2 the Klein-Goldberger model is used to illustrate these techniques.
First, construct the correlation matrix for the explanatory variables using the standardization in Equation 21.3.2.

```
xs = x[.,2 3 4];
xc = xs - meanc(xs)';
ssq = diag(xc'xc);
xc = xc ./ sqrt(ssq)';
corr = xc'xc;
corr;
```

Calculate the determinant of the correlation matrix and its inverse. The diagonal of the inverse correlation matrix contains the variance inflation factors.

```
detcorr = det(corr);
invcorr = inv(corr);
"det corr  : " detcorr;
" vif      : " diag(invcorr)';
```

Use PROC `OLS` to calculate the auxiliary regressions in Table 21.4. If these three lines are executed as a block, a touch to the space bar will let the next line to execute.

```
cls; {b1,cov} = myols(ones(t,1)~xs[.,2 3],xs[.,1]); wait;
cls; {b2,cov} = myols(ones(t,1)~xs[.,1 3],xs[.,2]); wait;
cls; {b3,cov} = myols(ones(t,1)~xs[.,1 2],xs[.,3]);
```

The R^2 values for the auxiliary regressions can also be obtained from the inverse of the correlation matrix.

```
auxr2 = 1 - (1 ./ diag(invcorr));
"aux R2  :"  auxr2';
```

To calculate Theil's multicollinearity effect regress y on the explanatory variables, deleting one at a time.

```
cls; {b1,cov} = myols(ones(t,1)~xs[.,1 2],y); wait;
cls; {b2,cov} = myols(ones(t,1)~xs[.,1 3],y); wait;
cls; {b3,cov} = myols(ones(t,1)~xs[.,2 3],y);
```

Calculate Theil's measure.

```
r2 = .9527;
Theil = r2 - (r2 - .9526) - (r2 - .9513) - (r2 - .8426);
"Theil's Measure"  Theil;
```

Write PROC `COLLIN` which calculates the characteristic roots and condition numbers of X'X in Table 21.5 as well as the table of variance proportions in Table 21.6. The argument X will be either the original X matrix or the normalized X matrix given by Equation 21.3.3. Place PROC `COLLIN` in a file and run it.

```
proc collin(x);
     local *;

/*  Calculate char. roots and vectors in descending order */

      {lam,p} = eigrs2(x'x);
      lam = rev(lam);
      p = (rev(p'))';

   /*  Calculate and print condition numbers  */

      ratio = lam[1,1] ./ lam;
      condno = sqrt(ratio);
      format 10,7;
      "Characteristic roots  " lam';
      "Ratio                 " ratio';
      "Condition numbers     " condno';

   /*  Calculate and print variance proportions   */

      prop = (p^2) ./ lam';
      propsum = sumc(prop');
      prop = prop ./ propsum ;
      ?; format 10,3;
      "Collinearity Diagnostic Table";
      prop';
```

```
        retp("");

endp;
```

Apply collinearity diagnostics to the original X matrix.

```
        collin(x);
```

Apply the collinearity diagnostics to the normalized X matrix.

```
        ssq = diag(x'x);
        xn = x ./ sqrt(ssq)';
        collin(xn);
```

21.4 Solutions to the Multicollinearity Problem

In this Section various ways of introducing nonsample information are offered as solutions to the problems caused by collinear data. The first is the use of exact restrictions. Use PROC `RLS` written in Chapter 20 to impose the Klein-Goldberger restrictions, discussed on page 876 of ITPE2, singly and jointly. Make sure PROC `RLS` is in memory or where **GAUSS** can find it. Compare your results to those in Table 21.7.
Use the first restriction alone.

```
        let r1[1,4] = 0 -.75 1 0;
        rr = 0;
        {br,covbr} = rls(x,y,R1,rr);
```

Use the second restriction alone.

```
        let r2[1,4] = 0 -.625 0 1;
        {br,covbr} = rls(x,y,R2,rr);
```

Use the two restrictions together.

```
        R = r1|r2;
        rr = zeros(2,1);
        {br,covbr} = rls(x,y,r,rr);
```

The use of stochastic restrictions in this chapter is somewhat different than in Chapter 20. The reason is that here it is not assumed that the error variance is known. The appropriate modification of Equation 20.2.15 is made in PROC `MIXED` given here. It takes as arguments `x`, `y`, `r`, `rr`, and `psi` and returns the mixed estimator and its estimated covariance matrix.

```
proc (2) = MIXED(x,y,R,rr,psi);
     local t,k,b,sse,sighat2,covbm,bm;

     t = rows(x);
     k = cols(x);
     b = y/x;
     sse = (y-x*b)'(y-x*b);
     sighat2 = sse/(t-k);

     covbm = invpd(x'x./sighat2 + R'*invpd(psi)*R);
     bm = covbm*(x'y./sighat2 + R'*invpd(psi)*rr);
     ?;
     "Mixed Estimation Results";
     ?;
     " R || rr" r~rr;
     " Est         :" bm';
     " Std. Err.   :" sqrt(diag(covbm))';
     retp(bm,covbm);
endp;
```

Use PROC `MIXED` first with prior variance 1/64.

```
psi = eye(2) ./ 64;
{bm,covbm} = MIXED(x,y,r,rr,psi);
```

Then using prior variance 1/256.

```
psi = eye(2) ./ 256;
{bm,covbm} = MIXED(x,y,r,rr,psi);
```

Compare your results to those in Table 21.8.
In Section 21.4.3 the ridge regression estimator is presented. Write PROC `HKBRIDGE` to compute the noniterative and iterative versions of the ridge estimator using the Hoerl-Kennard-Baldwin estimator of "k" given in Equation 21.4.12. The proc takes x and y in unstandardized form as arguments and returns the iterative estimates and estimated covariance matrix.

```
proc (2) = HKBRIDGE(x,y);
local *;
t = rows(x);                    /*  Standardize X and y  */
k = cols(x);
xs = x[.,2:k];
xc = xs - meanc(xs)';
ssq = diag(xc'xc);
std = sqrt(ssq);
xc = xc ./ std';
corr = xc'xc;
```

```
yc = y - meanc(y);

bc = yc/xc;                                  /*    OLS            */
sighat2 = (yc - xc*bc)'(yc - xc*bc)/(t-k);

khat = (k - 1)*sighat2 ./ (bc'bc);           /* Eq. 21.4.12       */
bridge = invpd(corr + khat .* eye(k-1)) * xc'yc;
                                             /* Eq. 21.4.13       */

q = invpd(corr + khat .* eye(k-1));
covridge = sighat2 .* q * corr * q;          /* Eq. 21.4.4        */

w = diagrv(eye(k-1),1 ./ std);               /*    W-inv          */
bridge = w*bridge;                           /* Unstandardize     */
covridge = w*covridge*w;
stderr = sqrt(diag(covridge));

"HKB-noniterative Ridge Estimator";          /*   Print           */

?;
" khat       : " khat;
" Est        : " bridge';
" Std. Err.  : " stderr';


crit = 1;                                    /*  define constants */
kold = khat;
iter = 1;

do until (crit le 1e-6) or (iter ge 20);     /*  begin loop       */
khat = (k-1) * sighat2 ./ (bridge'bridge);
crit = abs(khat - kold);
bridge = invpd(corr + khat .* eye(k-1)) * xc'yc;
kold = khat;
iter = iter + 1;
endo;

q = invpd(corr + khat .* eye(k-1));
covridge = sighat2 .* q * corr * q;          /*  Eq. 21.4.4       */

w = diagrv(eye(k-1),1 ./ std);               /*  Unstandardize    */
bridge = w*bridge;
covridge = w*covridge*w;
stderr = sqrt(diag(covridge));

?;
```

```
"HKB-iterative Ridge Estimator";          /*      Print        */
?;
" khat       : " khat;
" Est        : " bridge';
" Std. Err.  : " stderr';

retp(bridge,covridge);
endp;
```

Now use PROC HKBRIDGE to obtain the ridge regression estimates of the parameters, other than the intercept, as shown in Table 21.9.

```
{bridge,covridge} = HKBRIDGE(x,y);
```

Chapter 22

Robust Estimation

In this chapter estimation procedures are considered which are "robust" to changes in the data generation process. In particular stress is placed on robustness to nonnormal errors. A variety of regression diagnostics are introduced and illustrated.

22.1 The Consequences of Nonnormal Disturbances

In this section the properties of estimators under assumptions of finite and infinite error variances are summarized.

22.2 Regression Diagnostics

This section introduces the algebraic forms of a list of regression diagnostics. These will be illustrated in Section 22.4

22.3 Estimation Under Multivariate-t Errors

Multivariate-t errors are given as an example of nonnormal errors. The difference between independent and uncorrelated errors is noted and maximum likelihood estimation for the independent error case is described. This estimator is illustrated in the following Section.

22.4 Estimation Using Regression Quantiles

To introduce the concept of a regression quantile the simple model of the mean is used, Equation 22.4.1. Construct a vector y containing the data at the bottom of page 900 of ITPE2.

```
let y = 2 3 5 8 9 9 11 12 15;
```

Obtain an estimate of the 0.25 quantile as shown on the top of page 901.

```
t = rows(y);
theta = .25;
n = trunc(theta*t) + 1;
n;
est = y[n,1];
est;
```

To illustrate that the 0.25 quantile can also be defined as a solution to the minimization problem in Equation 22.4.3, calculate the objective function as the sum of two components as in Table 22.1 for various values of β. For each value of β the values of y that are greater than or equal to it are defined and the component of the objective function calculated. The results are accumulated in the matrix `store`. The objective function is minimized for $\beta = 5$.

```
store = zeros(9,4);
beta = 2;
do while beta le 10;
     select = y .>= beta;
     term1 = sumc(select .* (theta.* abs(y - beta)));
     term2 = sumc( (1-select) .* ( (1-theta) .* abs(y - beta)));
     fnval = term1 + term2;
     store[beta-1,.] = beta~term1~term2~fnval;
     beta = beta + 1;
endo;
"Table 22.1  ";
store;
```

To illustrate the use of regression diagnostics and robust estimation data based on independent t-errors, with 1 degree of freedom, is used. First, let us see what this p.d.f. looks like. The equation for the p.d.f. is given in Equations 22.5.2 and 22.5.3. Write a proc to compute the p.d.f. values given the values of ν and σ.

```
proc PDFT(e,nu,sig);
     local *;
     c1 = gamma( (nu+1)/2 ) * (nu*sig^2)^(nu/2) ./
          (gamma(.5)*gamma(nu/2));

     pdft = c1 .* (nu*(sig^2) + e.*e)^(-(1+nu)/2);
     retp(pdft);
endp;
```

Set the values of ν and σ, calculate the p.d.f. values of the t distribution and plot them.

```
nu = 1;
sig = 2;

e = seqa(-8,.02,801);
t = pdft(e,nu,sig);
library qgraph;
xy(e,t);
```

Note that the t-distribution with 1 degree of freedom is very flat and has thick tails. Thus the probability of getting large absolute errors is greater than with normal errors.
Table 22.2 in the text gives values of the dependent variable y generated from the model in Equation 22.5.1 with true parameter values $\beta_1 = 0$, $\beta_2 = 1$ and $\beta_3 = 1$, the given x values and independent t-errors. To construct the t-errors use equation 22.5.4 with z1 the first 40 observations from file NRANDOM.DAT and z2 the second group of 40 observations. Construct the errors and compare them to the last column of Table 22.2 in the text.

```
open f1 = nrandom.dat;
e = readr(f1,80);
f1 = close(f1);

z1 = e[1:40,.];
z2 = e[41:80,.];

e = (sig .* z1) ./ sqrt(z2^2);
e';
```

Now LOAD the data in file TABLE22.2 on the data disk, construct X and y and examine.

```
load dat[40,4] = table22.2;
t = 40;
x = ones(t,1)~dat[.,2 3];
let beta = 0 1 1;
y = x*beta + e;
y~x;
```

Use PROC MYOLS which was written in Chapter 6 to obtain least squares estimates of the parameters. Calculate estimated standard errors in the usual way, though we know that with t(1) errors the OLS estimator has neither mean nor variance. See Table 22.4.

```
{b,covb} = MYOLS(x,y);
```

The first regression diagnostic considered is teh Bera-Jarque test for normally distributed errors. The test statistic is given in Equation 22.2.7 and is based on the moment estimators described in Equation 22.2.6. Write a proc that will compute the test statistic and p-value given X and y.

```
proc BERAJAR(x,y);
     local *;

     t = rows(x);
     k = cols(x);
     b = y/x;                              /*  OLS        */
     e = y - x*b;                          /*  residuals  */
     sigtil2 = sumc(e^2)/t;                /*  moments    */
     mu3 = sumc(e^3)/t;
     mu4 = sumc(e^4)/t;
                                           /*  Eq. 22.2.7 */
     term1 = (mu3^2)/(6*sigtil2^3);
     term2 = (mu4 - (3 .* sigtil2^2))^2
                                      / (24 * sigtil2^4);
     lam = t*(term1 + term2);
     pval = cdfchic(lam,2);

     "  Bera-Jarque test for Normality";
     ?;
     "lam   " lam;
     "pval  " pval;
     retp("");
endp;
```

Use PROC BERAJAR to test the normality of the regression errors.

```
BERAJAR(x,y);
```

As noted, the hypothesis that the errors are normal is rejected at usual significance levels.
The other regression diagnostics illustrated are the leverage of an observation, studentized residuals, DFBETAS and DFFITS. Write a proc to calculate these values and print out a table, like Table 22.3 in the text, in which observations selected using the rule of thumb cutoffs (Section 22.2.2 and page 908) are displayed. This proc is long so place it in a convenient file and run it to store it into memory.

```
proc DIAGNOSE(x,y);
local t,k,b,h,obs,selh,sighat2,sigt2,estar,se,stid,c,dfbeta,
       selbeta,selfits,sel,table;

t = rows(x);
k = cols(x);
b = y/x;                                /*  OLS                */

/*  Calculate leverage values  */
```

```
h = diag(x*invpd(x'x)*x');              /*  Eq. 22.2.8          */
obs = seqa(1,1,t);
selh = (h .>= 2*k/t);                   /*  Find large h values */

/*  Calculate Studentized residuals  */

sighat2 = e'e/(t-k);

sigt2 = (t-k)*sighat2/(t-k-1) -         /*  Eq. 22.2.16         */
        ((e^2)/(t-k-1)) ./ (1-h);

estar = e ./ sqrt(sigt2 .* (1-h));      /*  Eq. 22.2.15         */
selstud = (abs(estar) .>= 2);           /*  Find large values   */

/*  Calculate DFBETAS   */

c = invpd(x'x)*x';
dfbeta = (c' .* (estar .*. ones(1,k)))       /*  Eq. 22.2.18   */
         ./ sqrt( (1-h) .*. diag(invpd(x'x))' );

dfbeta = dfbeta[.,2:k];
selbeta = abs(dfbeta) .>= (2/sqrt(t)); /*  Find large values   */

/*  Calculate DFFITS   */

dffits = sqrt(h ./ (1-h)) .* estar;      /*  Eq. 22.2.21       */
selfits = (abs(dffits) .>= 2*(sqrt(k/t)));/* Select lg. values */

/*  Find obs. with at least one significant diagnostic   */

sel = selh~selstud~selbeta~selfits;
sel = sumc(sel');
sel = sel .>= 1;
sel = selif(obs,sel);
table = sel~e[sel,.]~estar[sel,.]~h[sel,.]~
        dfbeta[sel,.]~dffits[sel,.];
table;
retp(table);
endp;
```

Use PROC `DIAGNOSE` to print out a table like Table 22.3 in the text. It will not be identical due to the fact that the table in the text identifies other observations as well.

```
table223 = diagnose(x,y);
```

Given that nonnormal errors have been identified the use of robust estimators may be called for. Regression quantiles and trimmed least squares use linear programming techniques which will not be covered here. Instead we will use the maximum likelihood estimation technique, under the assumption of independent t-errors as described in Section 22.3. The maximum likelihood estimates satisfy the four equations 22.3.9 to 22.3.12. To obtain estimates that satisfy those constraints we will iterate using the OLS estimates as starting values.

```
bmle = b;
crit = 1;
nu = 1;
iter = 1;
do until (crit le 1.e-6) or (iter ge 25);
emle = y - x*bmle;
sig2mle = emle'emle/t;
nusig2 = (nu + 1) * sig2mle;         /*  Eq. 22.3.12 */
wt = 1 ./ ( 1 + (emle^2)./nusig2 );  /*  Eq. 22.3.11 */
wx = wt .* x;                        /*  Weight obs  */
wy = wt .* y;
newb = invpd(x'wx)*x'wy;             /*  Eq. 22.3.9  */
crit = maxc(abs(bmle-newb));
bmle = newb;
iter = iter + 1;
iter~crit~bmle';
endo;
```

Calculate the asymptotic covariance matrix as given in Equation 22.3.13 and print the results.

```
covml = ((nu + 3)*nusig2 / (nu+1)) * invpd(x'x);
se = sqrt(diag(covml));
"bmle   " bmle';
"se     " se';
```

Appendix A

Linear Algebra and Matrix Methods

In this section the use of **GAUSS** matrix operations is illustrated. These operations are used throughout the text.

A.1 Definition of Matrices and Vectors

A matrix is a rectangular array. A vector is a single row or column. When defining a matrix in **GAUSS** using the LET command the matrix is assumed to be a vector unless dimensions are given. For example

```
let x[5,1] = 3 9 0 -5 1;
x;
```

or

```
let x = 3 9 0 -5 1;
x;
```

The transpose of a column vector is a row vector.

```
x';
```

An identity matrix is specified as

```
I = eye(3);
I;
```

The diagonal of a square matrix can be extracted as

```
ivec = diag(I);
ivec';
```

The diagonal of a square matrix can be inserted as

```
let dvec = 1 2 3;
d = diagrv(I,dvec);
d;
```

Matrix equality requires that two matrices be equal element by element, which means they must be of the same dimension. The equality symbol (=) in **GAUSS** means a bit more however. It is truly an "assigment" operator, where the symbol on the left-hand-side is replaced by whatever is on the right-hand-side. Thus for example,

```
d = sumc(d);
d;
```

is legal and simply means that the symbol `d` is assigned the value of `sumc(d)`, where SUMC sums the elements of the the columns of its matrix argument. Matrix equality can be checked using the relational operator == (or EQ) which returns a value of 1 or 0 depending upon whether two matrices are equal element by element. See the **GAUSS** manual for further explanation of RELATIONAL OPERATORS, ELEMENTWISE OPERATORS and LOGICAL OPERATORS.

A.2 Matrix Addition and Subtraction

Formally, matrix addition and subtraction requires that the matrices involved be of the same dimension.

```
let A[2,2] = 6  8
            -2  4;

let B[2,2] = 0 -3
             9  7;
C = A + B;
C;
```

GAUSS commands are somewhat more flexible in that it offers "elementwise" operations. To add or subtract a constant to every element of a matrix.

```
C = 5 + A;
C;
```

To add a row to each row.

```
let d[1,2] = 9 1;
C = A + d;
C;
```

To add a column to each column.

```
let d = 2 3;
C = A + d;
C;
```

A.3 Matrix Multiplication

Matrices must be conformable for multiplication.

```
let D[2,3] = 3  -8  1
             9   2  5;
F = A*D;
F;
```

Inner products of vectors and matrices can be shortened.

```
F = A'*A;
F;
```

or

```
F = A'A;
F;
```

Scalar multiplication is defined as

```
F = 5*A;
F;
```

Elementwise multiplication (.*) can be used to multiply corresponding elements, corresponding rows or columns.

```
/*  Multiply corresponding elements           */
F = A .* B;
F;

/*  Elementwise multiply each row of A by d   */
let d[1,2] = .5 1;
F = A .* d;
F;

/*  Elementwise multiply each column of A by e */
let e = 3 2;
F = A .* e;

F;
```

Elementwise divison (./) works in the same way. Note that A ./ B would result in a divison by zero, which is to be avoided.

```
F = A ./ d;
F;

F = A ./ e;
F;
```

Another useful elementwise operater is exponentiation, (^).

```
F = A^2;
F;
```

A.4 Trace of a Square Matrix

GAUSS does not have a trace operator, thus to sum diagonal elements combine SUMC and DIAG as

```
tr = sumc(diag(A));
tr;
```

A.5 Determinant of a Square matrix

The **GAUSS** determinant function is DET.

```
c = det(A);
c;
```

A.6 The Rank of a Matrix and Linear Dependency

The rank of a matrix is the number of linearly independent rows or columns. A square matrix with full rank (i.e., rank equals dimension) is nonsingular and has an inverse and a nonzero determinant. **GAUSS** has a function, RANK, which computes the rank of a matrix by counting its number of nonzero "singular values". More is given in Section A.11.

A.7 Inverse Matrix and Generalized Inverse

The inverse of a square nonsingular matrix can be found in several ways in **GAUSS**. The relevant functions are INV, INVPD and SOLPD. Use INV to find the matrix inverse of a general, nonsingular matrix.

```
inva = inv(A);
inva;
check = inva*A;
check;
```

If the matrix is positive definite (Section A.14) and symmetric then the function INVPD can be used and is faster than INV.

```
/*   AA is a positive definite and symmetric matrix  */
AA = A'A;
AA;
aainv = invpd(AA);
aainv;
check = aainv*AA;
check;
```

The function SOLPD will be explained in the following section.
GAUSS does provide a command to find the generalized (Moore-Penrose) inverse, PINV.

A.8 Solutions for Systems of Simultaneous Linear Equations

To solve a system of equations like Equation A.8.2 the matrix A must be nonsingular. If that is true then the matrix inverse function can be used to solve for the unknowns x. Using the matrix `A` already in memory,

```
let b = 2 1;
x = inva*b;
check = A*x;
check;
```

CHECK is identical to the vector B as x is the solution to the system of equations and thus satisfies the equality.
If `A` is positive definite then the faster function SOLPD can be used. Using `AA` created just above for `A`,

```
x = solpd(b,AA);
check = AA*x;
check;
```

SOLPD can also be used to invert a positive definite matrix.

```
aainv = solpd(eye(2),AA);
check = aainv*AA;
check;
```

A.9 Characteristic Roots and Vectors of a Square Matrix

The characteristic roots and vectors of a square matrix are found by solving the characteristic polynomial (A.9.2) for lambda and then finding vectors x which solve (A.9.1). If the square matrix is symmetric then the characteristic roots are real. Otherwise they may not be. **GAUSS** provides functions that finds the characteristic roots and vectors for both of these cases.
First, for symmetric matrices use the functions EIGRS (roots only) or EIGRS2 (roots and vectors). For example, using the matrix `AA`.

```
{r,v} = eigrs2(AA);
r~v;
```

Note that **GAUSS** returns the roots in ascending order of magnitude. The Characteristic vectors are found in the columns of V and are in the order of the characteristic roots. Check that the first characteristic root and vector solve (A.9.1).

```
c = (AA - r[1,1]*eye(2))*v[.,1];
c;
```

If the square matrix is not symmetric use the functions EIGRG or EIGRG2. These functions return the real and complex parts of the roots and vectors. Using the example in the text,

```
let A[2,2] = 5  -3
             4  -2;
{rr,ri,vr,vi} = eigrg2(A);
rr~ri;
vr~vi;
```

Note that in this example the roots are real and returned in descending order. The characteristic vectors are real as well. As the text notes, characteristic vectors are not unique with respect to sign, and in this case **GAUSS** returns the the positive ones.

A.10 Orthogonal Matrices

An orthogonal matrix is one whose transpose is its inverse. As noted in the text, the characteristic vectors of a symmetric matrix form an orthogonal set. The matrix V containing the characteristic vectors of the symmetric matrix AA is an example.

```
I1 = v'v;
I1;
```

```
I2 = v*v';
I2;
```

Furthermore, `AA*v = v*diag(lambda)` where `diag(lambda)` is a diagonal matrix with the characteristic roots on the diagonal. To illustrate,

```
m1 = AA*v;
m2 = v*diagrv(eye(2),r);
m1~m2;
```

A.11 Diagonalization of a Symmetric Matrix

Using the results in A.10 and A.11 we can construct a matrix P that "diagonalizes" a positive definite and symmetric matrix. First, note that the matrix of characteristic vectors diagonalizes a symmetric matrix and returns a diagonal matrix with the characteristic roots on the diagonal.

```
lam = v'*AA*v;
lam~r;
```

We can further diagonalize a matrix to an identity by using the matrix V post-multiplied by a diagonal matrix whose nonzero elements are the reciprocals of the square roots of the characteristic roots.

```
P = V*diagrv(eye(2),1 ./ sqrt(r));
W = P'*AA*P;
W;
```

It is useful to know that the rank of a matrix A is given by the number of nonzero characteristic roots of A'A. In **GAUSS** the rank of a matrix is calculated from the number of nonzero "singular values" of A, which are in fact the square roots of the characteristic roots of A'A. The function that does this is RANK. The command that performs the singular value decomposition is SVD.
A related transformation is the Cholesky decomposition, CHOL, which factors a matrix (X) into the product X = Y'Y where Y is upper triangular.

A.12 Idempotent Matrices

An idempotent matrix is one which when multiplied by itself yields itself again. To illustrate an econometricians favorite idempotent matrix

```
let X[5,2] = 1 5
             1 7
             1 3
             1 1
```

```
                1 0;

M = eye(5) - X*invpd(X'X)*X';
```

The matrix M is symmetric and idempotent.

```
Q = M*M;
format 8,4;
Q;
?;
M;
```

The characteristic roots of a symmetric, idempotent matrix are ones and zeros with the number of unit roots being its rank.

```
lam = eigrs(M);
lam;
```

Furthermore the trace of an idempotent matrix is also equal to its rank.

```
sumc(diag(M));
```

A.13 Quadratic Forms

A quadratic form in x is defined in (A.13.1). The sign of the quadratic form is related to the characteristics of the matrix, A, as is illustrated in the following section.

A.14 Definite Matrices

Definite matrices have many useful properties as listed here. These will be used frequently throughout the text.

A.15 Kronecker Product of Matrices

The Kronecker product of two matrices is defined in (A.15.1). The **GAUSS** operator .*. yields this product.

```
Let A[2,2] = 1 3
             2 0;
Let B[2,3] = 2 2 0
             1 0 3;

C = A .*. B;
C;
```

Properties of this operator are listed in the text. Particularly useful is Equation A.15.6.

A.16 Vectorization of Matrices

Vectorization of a matrix means that its columns are "stacked" one atop the other as in (A.16.1). In **GAUSS** this can be accomplished using the RESHAPE function. For example, use the matrix B in the previous section.

```
vecb = reshape(B',6,1);
vecb;
```

Note that the transpose of B was reshaped as this function reshapes the rows of a matrix. Alternatively the GAUSS function VEC can be used.

```
vecb = vec(B);
vecb;
```

A.17 Vector and Matrix Differentiation

Useful rules for differentiation are given in this section. Take particular care to understand the definitions of gradient, Jacobian and Hessian.

A.18 Normal Vectors and Multivariate Normal Distribution

Transformations of multivariate normal random vectors play a large role in statistics and are related to hypothesis testing and confidence interval estimation in this book. Especially useful is Equation A.18.6 which gives the distribution of linear transformations of multivariate normal random vectors.

A.19 Linear, Quadratic, and Other Nonlinear Functions of Normal Random Vectors

This Section extends the results in A.18 to quadratic and linear forms in multivariate normal random vectors. Chisquare, Student's-t and the F- distribution are defined and related. The gamma function is given by the **GAUSS** function GAMMA. **GAUSS** also includes functions for the p.d.f. and c.d.f. of N(0,1) random variables (PDFN & CDFN) as well as the complement of the normal c.d.f. (CDFNC). Also provided, and useful for finding "p-values" are complements to the distribution functions of the t (CDFTC), F (CDFFC) and chi-square (CDFCC).

A.20 Summation and Product Operators

GAUSS provides functions that sum (SUMC) and multiply (PRODC) columns of matrices. The factorial operator is the exclamation symbol "!".