Gauss Programs for Bayesian Inference and Markov Chain Monte Carlo

November 2001

Peter Lenk

Peter Lenk is Associate Professor of Statistics and Marketing, The University of Michigan Business School, Ann Arbor, MI 48109-1234, Phone: 734–936–2619 and Fax: 734–936–0274, Emai: plenk@umich.edu

Contents

Chapter 1

Getting Started with GAUSS

Outline

1. **What is Gauss?**
2. **Scalars, Vectors, and Matrices**
3. **Operators**
4. **Matrix Functions**
5. **Files, Input & Output**

1.1 What is Gauss?

Gauss is a matrix based language with many built–in functions. The basic data structures in Gauss are matrices. Special cases are vectors and scalars. Operators are matrix operators. It also has a full range of graphical features and random number generators.

1.2 Scalars, Vectors, and Matrices

1. **Define scalars:**

$$c = 4; d = 2;$$

2. **Define a row vector:**

$$x = \textbf{\{ 1 2 3 4 \}};.$$

Type "print x;"

3. **Define a column vector:**

$$x = \textbf{\{ 1, 2, 3, 4\}} \ ;$$

Type "print x;"

4. **Do you ever need a sequence of numbers?**

$$x = \textbf{seqa}(1, d, c);$$

5. **Define a matrix (entered by rows):**

$$amat = \textbf{\{ 1 2 3, 4 5 6, 7 8 9, 10 11 12\}};$$

Type "print amat;"

6. Do you ever need a matrix of zeros

$$cmat = \mathbf{zeros}(5, 3);$$

or ones?

$$dmat = \mathbf{ones}(2, 4);$$

What about the identity matrix:

$$idmat = \mathbf{eye}(c);$$

7. You can get submatrices:

$$ar1 = amat[1, .]; \ \textbf{@Row 1@}$$

$$ar24 = amat[2 : 4, .]; \ \textbf{@ Rows 2 thru 4 @}$$

$$ac2 = amat[., 2]; \ \textbf{@ Column 2 @}$$

$$ac23 = amat[., 2\ 3]; \ \textbf{@ Columns 2 and 3 @}$$

$$asub = amat[2 : 4, 2 : 3]; \ \textbf{@ What is it? @}$$

1.3 Operators

1. **Matrix multiplication is simple:**

$$bmat = \{ \mathbf{20\ 21,\ 22\ 23,\ 24\ 25} \};$$
$$cmat = amat * bmat;$$

2. **Matrix transpose is:** $amatt = amat';$

3. **How about:** $dmat = x'amat;$

4. **Try** $z = x + amat;$

5. **Here is something different:**

   $y = x.*amat;$ **@ Element by element multiplication @**

6. **Try this out:**

$$z = x. * .amat;$$ **@ Kronecker product @**

1.4 Matrix Functions

1. **Suppose you have a positive definite covariance matrix:**

$$sigma = \mathbf{\{\ 2\ .8\ -.5,\ .8\ 1\ -.3,\ -.5\ -.3\ .7\ \}};$$

2. **You can invert it with:**

$$sigmai = \mathbf{invpd}(sigma);$$

Check it with:

$$a = sigma * sigmai;$$

3. **Its determinate is**

$$detsig = \mathbf{det}(sigma);$$

4. **Its Cholesky decomposition:**

$$sig12 = \mathbf{chol}(sigma);$$

Check

$$a = sig12' sig12$$

5. **We use this in generating** $N(y|\mu, \Sigma)$**:**

$$\begin{aligned} mu &= \{1, 5, 10\}; \\ y &= mu + sig12' \mathbf{rndn}(3, 1); \end{aligned}$$

Cool! Note that rndn(3,1) gives a vector of independent, standard normals. rndn(3,10) gives a 3 by 10 matrix of standard normals.

6. Try

$$
\begin{aligned}
y &= mu' + \mathbf{rndn}(100, 3) * sig12; \\
ybar &= \mathbf{meanc}(y); \\
ystd &= \mathbf{stdc}(y); \\
ycor &= \mathbf{corrx}(y); \\
\{f, m, p\} &= \mathbf{hist}(y[., 1], 10); \\
\mathbf{library} &\quad \mathbf{pgraph;} \\
\{f, m, p\} &= \mathbf{hist}(y[., 1], 10); \\
\mathbf{xy(y[.,1],y[.,2]);} & \\
\mathbf{\_plctrl} &= -1; \\
\mathbf{xy(y[.,1],y[.,2]);} & \\
\mathbf{graphset;} &
\end{aligned}
$$

1.5 Files, Input & Output

1. ASCI Files & Simple Data Files

- **Read ASCI File**

```
load xdata[10,3] = "c:\\mcmc\\inout\\data0.txt";
@ data0.txt has 10 rows and 3 columns   @
@ data0.txt does not have variable names @
@ xdata is a 10 by 3 matrix in Gauss     @
print xdata;
```

- **Save a matrix from GAUSS.**

```
save ydata = xdata;
@ Save xdata into file ydata.fmt @
```

- **Load a .FMT File.**

```
load zdata = ydata;
@ load file ydata into zdata  @
```

2. GAUSS Files:

• Creating a GAUSS data file.

```
new; @ Clear memory @
@ Generate some data @
nobs = 100; @ number of observations  @
nvars = 5; @ number of variables @
vdata = rndn(100,5); @ 100 by 5 matrix of standard normals @

@ Character vector of variables names @
let vnames = var1  var2   var3   var4   var5;

@ Create the Gauss file vdatag @
create f1 = vdatag with ^vnames, 0, 8;
@ The data is written to vdatag.dat   @
@ Names are in the vdatag.dht  @
@ f1 is the file handle   @

@ Write the matrix vdata to gauss file vdatag @
wnum = writer(f1,vdata);
@ wnum = number of observations written to f1@

@ Close the file @
cok = close(f1);
@ cok =  0 if successful; = -1 if unsuccessful @
```

- **Reading a GAUSS file.**

```
new; @ Clears memory @
infile  = "vdatag"; @ Name of Gauss File @
@ infile is the input 'file handle' for vdatag @

@ Input Gauss files @
@ Opens Gauss file & assigns file handle f1 @
open f1  = ^infile;
vdata = readr(f1,rowsf(f1));
@ vdata is the name of the matrix  @
@ readr reads Gauss file with file handle f1. @
@ rowsf(f1) returns the number of rows in the Gauss file.  @
@ readr reads rowsf(f1) rows, which is the entire dataset. @
cok = close(f1); @ close the file @

@ Get the variable names @
vnames = setvars(infile);

@ Print the variable names  @
@ $ indicates a character vector  @
print $ vnames;
```

3. EXCEL Files.

- **Import EXCEL, version 4.0.**

  Convert it to a GAUSS file.

```
@ Example of converting EXCEL file to a gauss file @
new;
@ Set file names and paths @
fname = "c:\\mcmc\\InOut\\data2.xls";
@ EXCEL version 4.0 First row has variable names. @
dname = "c:\\mcmc\\InOut\\data2g.dat";
@ GAUSS data file   @
iok  = importf(fname, dname,0,1);
@ Convert EXCEL file to GAUSS data file @
@ iok = 1 if success and = 0 if failure @



@ Read Gauss File @
infile  = "data2g";
open f1  = ^infile;
vdata2 = readr(f1,rowsf(f1));
cok = close(f1);
vn = setvars(infile);
print $ vn;
```

- **Export Gauss file to EXCEL file.**

```
@ Export Gauss data file to EXCEL @
fold = "c:\\mcmc\\InOut\\data2g.dat";
fnew = "c:\\mcmc\\InOut\\data3.xls";
exok = exportf(fold, fnew ,0);
@ fold is the path & name of the Gauss file @
@ fnew is the path & name of the target EXCEL file @
@ 0 is default -> all variables @
@ exok = 1 is success; = 0 is failure @
```

- **Export Data to EXCEL File**

```
@ Export data to EXCEL file @
@ Create some data. @
mdata = rndn(50,4);

@ Give variables their names @
let mnames = bayes is more fun;

@ Give path + file name for EXCEL file @
outxls = "c:\\mcmc\\InOut\\mdata.xls";

@ Export it. @
exok = export(mdata, outxls, mnames);
```

Chapter 2

GAUSS Libraries

GAUSS allows user defined subroutines to be added to the GAUSS programing environment. Extending Gauss is accomplished with libraries files. Libraries define collections of subroutines or Gauss procedures. These procedures are complied by Gauss with the command:

$$\text{library} < lib1 >, < lib2 >, \ldots < libn >;$$

The graphics library is "pgraph." Before using graphics, type:

```
library pgraph;
```

We used this command on page (10).

Library files define procedures for the GAUSS compiler. They have an "lcg" extension and reside in the subdirectory:

```
C:\GAUSS\LIB
```

My programs uses the library bamm.lcg, which contains my subroutines for Simpson's integration and some random number generators. Before running my programs, you need to issue:

library pgraph, bamm;

bamm.lcg only defines subroutines contained in one file. In general, one LCG library can pull subroutines from several files, and one file of procedures can be used by more than one LCG file. The only constraint is that on a particular library call, each subroutine is uniquely defined by a LCG file. The contents of bamm.lcg is:

```
c:\mcmc\src\bamm.src
      intsim                          : proc
      dir1                            : proc
      dirord                          : proc
      wishart0                        : proc
      wishart                         : proc
      rndmn                           : proc
      rndzmn                          : proc
      rndtna                          : proc
      rndtnb                          : proc
      rndnab                          : proc
      rndgamlo                        : proc
      rndgamup                        : proc
```

The first line gives the path and file name for the file that contains the procedures. The following lines gives names of the procedures that you wish to compile. You do not have to list all procedures in your file.

2.1 SRC Files

The code for the procedures in a library file is contained in a SRC file. These have the extension "src" and are in the directory. GAUSS's SRC files are in:

```
C:\GAUSS\SRC
```

However, you can place your procedures anywhere, just as long as the LCG gives the correct path. Each procedure in the SRC file has to be defined in a LCG file. bamm.lcg and bamm.src work together.

```
/*
*************************************************************************
*   (C) Copyright 1999, Peter Lenk. All Rights Reserved.
*  BAMM.SRC are my procedures for Lenk's MCMC
*     INTSIM            performs Simpson's integration.
*                       Note: Gauss does it too, but my version is simpler when
*                       fx is computed over a fixed grid x to plot fx versus x.
*     DIR1              generates a dirichlet random deviate.
*     DIRORD            generates ordered dirichlet random deviates
*     WISHART0          generates a standard Wishart.
*     WISHART           generates general Wishart & Inverse Wishart
*     RNDMN             generates multinomial counts.
*     RNDZMN            generates indicator for bin gets ball.
*     RNDTNA            generates truncated normals above a value
*     RNDTNB            generates truncated normals below a value
*     RNDNAB            generates normal between a and b.
*     RNDGAMLO          generates truncated gamma above lower bound.
*************************************************************************
*/

/*
*********************************************************
*  INTSIM performs simpson's integration
*  f = function, grid must have 2*m+1 points, k cols
*  del = grid size, scalar or k by 1 vector
*********************************************************
*/
PROC intsim(f,delta);
local  r, t, even, fint;
fint = 0;
r = rows(f);
if r == 2*floor(r/2);
print "ERROR: Even number of rows for simpson's integration";
elseif r == 3; @ Simple Simpson's Integration @
t = 1|4|1;
fint = (t'f).*(delta')/3;
else; @ Composite Simpson's Integration @
t = 1|(ones((r-3)/2,1).*.(4|2))|4|1;
fint = (t'f).*(delta')/3;
endif;
retp(fint');
endp;
```

```
/*
************************************************************
*  DIR1.G generates a matrix of Dirichlet random variables
*  Rows are independent random variables.
*  INPUT
*     xi = n x h matrix of parameters
*     Each row are the parameters for a Dirichlet.
*  OUTPUT
*     x  =  n by h matrix of dirichlet probabilities
*           each row is an independent dirichlet(xi).
************************************************************
*/
PROC dir1(xi);
local x,r,c;
r = rows(xi);
c = cols(xi);
x = rndgam(r,c,xi);
retp(x./sumc(x'));
endp;

/*
*********************************************************************
*  DIRORD generates truncated dirichet random deviates.
*  p_1 <= p_2 <= ... <= p_K
*  Uses slice sampling.
*  INPUT:
* alpha = k x 1 vector of "alpha" values
* xgam = n x k vector of truncated gammas used in prob
*  OUTPUT:
* prob = n x k matrix of probabilities
* xgam = updated values of truncated gammas.
*********************************************************************
*/

PROC (2) = dirord(alpha,xgam);
local i,k,n,u1,u2,umin,c;
k = cols(xgam);
n = rows(xgam);
u1 = rndu(n,k);
u2 = rndu(n,k);

if alpha[1] == 1; @ alpha = 1 implies that x is exponential @
xgam[.,1] = -ln(1-u2[.,1]);
else;
c = 1/(alpha[1]-1);
```

```
umin  = (u1[.,1]^c).*xgam[.,1];
umin = (umin .< 0).*(0-umin) + umin;  @ umin > 0 @
xgam[.,1] = umin - ln(1-u2[.,1]);
endif;
i = 2;
do while i <= k;
if alpha[i] == 1; @ alpha = 1  @
umin        = xgam[.,i-1];
xgam[.,i] = umin-ln(1-u2[.,i]);
else;
c = 1/(alpha[i]-1);
umin  = (u1[.,i]^c).*xgam[.,i];
umin = maxc( (umin~xgam[.,i-1])' );
umin = (umin .< 0).*(0-umin) + umin; @ umin > 0 @
xgam[.,i] = umin - ln(1-u2[.,i]);
endif;
i = i + 1;
endo;
retp(xgam./(sumc(xgam')), xgam);
endp;


/*
************************************************************
*  WISHART0.G generates a random deviate from a standard
*  Wishart distribution.
*  m = dimension, n = df.
*  Bartletts decomposition: W = T*T' where T upper diagonal
*  of T is zero.  Lower diagonal elements of T are normal.
*   Diagonal elements of T are sqrt(chi-square).
*   See page 99 of Ripley, Stochastic Simulation.
************************************************************
*/
PROC wishart0(m,n);
local  i,j,t,x,alpha;
t = zeros(m,m);
if m >= n;
print "ERROR: DF of Wishart < Dimension";
elseif m > 1; @ more than one dimension  @
@ Create low diagonal matrix with N(0,1) deviates @
t = rndn(m*(m-1)/2,1); @ vector of normals  @
t = lowmat(xpnd(t)); @ expand into symmetric & reshape into lower diagonal matrix @
t = zeros(1,m)|(t~zeros(m-1,1)); @ fill-out zeros @
@ put gamma down diagonal    @
alpha = seqa(n,-1,m)/2;
```

```
x = sqrt( rndgam(m,1,alpha)*2 );   @ gamma(alpha,0.5) @
t = diagrv(t,x);
retp(t*t'); @ outer product gives bartlett's decomposion @
elseif m == 1; @ m = 1 -> Wishart is gamma(df/2,1)  @
x = rndgam(1,1,n/2);
retp(x);
else;
print "ERROR: DF of Wishart < 1:  df = " m;
endif;
endp;

/*
**************************************************************************
* WISHART
* Generates general Wishart and Inverted Wishart
* INPUT
* dim = dimension of matrix
* df = degree of freedom
* gmat = scale matrix
* OUTPUT
* wish = random matrix with the Wishart distribution
* wishi = random matrix that is inverted Wishart.
* NOTE:
* In most MCMC applications,
* df is the posterior degrees of freedom:
* df = df0 + n where df0 = prior df.
* gmat is the posterior scale matrix:
* gmat = (g0^{-1} + sum_{i=1}^n (y_i - mu_i)(y_i - mu_i)' )^{-1}
* where g0 is the prior scale matrix.
* wishi  = Sigma, the covariance matrix
* wish  = Sigma^{-1}
**************************************************************************
*/
PROC (2) = wishart(dim,df,gmat);
local wish, wishi, gmat12, w;
gmat12 = chol(gmat);
w = wishart0(dim,df); @ Get standard Wishart  @
wish = gmat12'w*gmat12; @ Wishart @
wishi = invpd(wish); @ Inverted Wishart @
retp(wish,wishi);
endp;

/*
**********************************************************
*  RNDMN generates multinomial random deviates
```

```
*    X is MN_K(n,p): n trials, k cells, p = probs.
*
*  Input:
*         n = number of trails
*         p = j by k matrix of probabilities
*             Each row is a probability vector
*
*  Output:
*         X = j by k matrix
*             Each row is multinomial counts.
*
************************************************************
*/
PROC rndmn(n,prob);
local x,xrows,cells,i,u;
xrows = rows(prob);
cells = cols(prob);
if not ((prob >= 0 ) and (prob <= 1 ));
   errorlog "ERROR in RNDMN.G: probability not between 0 and 1";
endif;
if not sumc(prob') == 1;
   errorlog "ERROR in RNDMN.G: probability does not sum to 1";
endif;
x     = zeros(xrows,cells);
i     = 1;
u = rndu(xrows,n);
do while i <= cells;
   x[.,i] = sumc(( (u.>0) .and (u.<= prob[.,i]))');
   u = u - prob[.,i];
   i = i + 1;
endo;
retp(x);
endp;

/*
************************************************************
*  RNDZMN.G generates multinomial(1,prob).
*  Z indicates the bin that gets the ball.
*  INPUT
*     prob = n x h; n = number of obs, h = number of bins.
*     prob[i,j] = P(z_i is in bin j)
*     rows are independent.
*  OUTPUT
*     z = n x 1 vector
*     z[i] = j if bin j received ball.
```

```
*************************************************************
*/
PROC rndzmn(prob);
local z,j,u,h,n;
n = rows(prob);
h = cols(prob);
if not ((prob >= 0 ) and (prob <= 1 ));
   errorlog "ERROR in RNDZMN.G: probability not between 0 and 1";
endif;
if not (( sumc(prob') > 0.999999) and (sumc(prob') < 1.000001));
   errorlog "ERROR in RNDZMN.G: probability does not sum to 1";
endif;
z = zeros(n,1);
u = rndu(n,1);
j = 1;
do while j <= h;
   z = z + ((u.>0).and (u.<= prob[.,j])).*j;
   u = u - prob[.,j];
   j = j + 1;
endo;
retp(z);
endp;

/*
**************************************************************************
*   RNDTNA generates form a truncated normal distribution
* Truncation is above from (-infinity, xtop)
*    INPUT
* mu = vector of means
* sigma = vector of stds
* xtop = upper limit
*   OUTPUT
* x = truncated normal
**************************************************************************
*/
PROC rndtna(mu,sigma, xtop);
local   u, fb, x ;
u = rndu(rows(mu),1);
fb = cdfn( (xtop - mu)./sigma);
x = mu + sigma.*cdfni( u.*fb );
x = (xtop - x).*(x .> xtop) + x;
retp(x);
endp;
```

```
/*
**************************************************************************
*   RNDTNB generates form a truncated normal distribution
*   Truncated on [xbot, infinity)
*    INPUT
* mu = vector of means
* sigma = vector of stds
* xbot = lower limit
*   OUTPUT
* x = truncated normal
**************************************************************************
*/

PROC  rndtnb(mu,sigma,xbot);
local   u, fa, fb, x ;
u = rndu(rows(mu),1);
fa = cdfn( (xbot - mu)./sigma);
fb = 1;
x = mu + sigma.*cdfni(fa + u.*(fb-fa));
x = (xbot - x).*(x .< xbot) + x;
retp(x);
endp;

/*
**************************************************************************
*  RNDNAB
* Generate a matrix of normals that are constrained between two numbers
* f(x) \propto I(low < x < up)*N(x|xm, xv)
*
* INPUT:
* nr = number of rows
* nc = number of columns
* xm = mean
* xv = variance
* low = lower bound
* up = upper bound
* OUPUT
* nr by nc matrix  of random numbers
**************************************************************************
*/
proc  rndnab(nr,nc,xm,xv,low,up);
local flow, fup, xstd, x;
if low > up;
errorlog "Error in RNDNAB: Lower bound exceeds upper bound.";
print "low = " low " up = " up;
```

```
retp(0);
endif;
xstd = sqrt(xv); @ STD of X @
flow = cdfn((low-xm)/xstd); @ Normal cdf of lower bound @
fup = cdfn((up-xm)/xstd); @ Normal cdf of upper bound @
if flow >= 1;
errorlog "RNDNAB: F(lower bound) = 1";
retp(0);
endif;
if fup <= 0;
errorlog "RNDNAB: F(uppper bound) = 0";
retp(0);
endif;

x = xm + xstd*cdfni( (fup - flow)*rndu(nr,nc) + flow) ;
x = (low - x).*(x .< low) + x;
x = (up - x).*(x .> up) + x;
retp(x);
endp;


/*
****************************************************************
* RNDGAMLO.G
*
* f(x) \propto I( x> low) x^{alpha-1}exp(-beta*x)
*
* INPUT:
* nr = number of rows
* nc = number of columns
* alpha = shape parameters of gamma: scalar
* beta = scale parameter: scalar
* low = lower limit: scalar.
*
* OUTPUT
* truncated gamma
****************************************************************
*/
proc  rndgamlo(nr, nc, alpha,beta,low);
local glow, p, a2, x;
a2 = alpha*ones(nr,nc);
glow = cdfgam(alpha,beta*low);
if glow >= 1; glow = 0.99999; endif;
p = rndu(nr,nc).*(1-glow) + glow;
x = gammaii(a2, p)./beta;
```

```
x = (low - x).*(x .< low) + x;
retp(x);
endp;


/*
****************************************************************
* RNDGAMUP.G
*
* f(x) \propto I( x < up) x^{alpha-1}exp(-beta*x)
*
* INPUT:
* nr = number of rows
* nc = number of columns
* alpha = shape parameters of gamma: vector
* beta = scale parameter: vector
* low = lower limit: scalar.
*
* OUTPUT
* truncated gamma
****************************************************************
*/
proc  rndgamup(nr, nc, alpha,beta,up);
local gup, p, a2, x;
a2 = alpha.*ones(nr,nc);
gup  = cdfgam(alpha,beta*up);
if gup >= 1; gup = 0.99999; endif;
p = rndu(nr,nc).*gup;
x = gammaii(a2, p)./beta;
x = (up - x).*(x .> up) + x;
retp(x);
endp;
```

Chapter 3

Linear Regression

3.1 Data Generation

```
/*
**************************************************
*   GREG1.GSS
* Generates data for linear regression
**************************************************
*/
new;
nobs = 30; @ number of observations  @
@ Define true regression parametes @
betat  = {
 2,
-1,
 3,
 0
   };
@ Enter vector or matrices by rows, separated by commas  @
sigmat = 2; @ true error std @
rankx = rows(betat); @ rows(x) = number of rows for x  @
xdim = rankx-1; @ number of dependent variables  @
xnames = 0 $+ "X" $+ ftocv(seqa(1,1,xdim),1,0);
@ Makes a character string that is X1, X2, ... @
ynames = "Y";
xynames = xnames|ynames;



xdata = rndn(nobs,xdim); @ rndn -> N(0,1) random numbers @
xmat = ones(nobs,1)~xdata; @ design matrix @
@ ~  pastes two matrics side-by-side @

@ Generate depenent observations @
ydata = xmat*betat + sigmat*rndn(nobs,1);
@ rndn(r,c) generates a r x c matric of N(0,1) random deviates  @

xydata = xdata~ydata;

/*
**************************************************
* Create a Gauss file.  f1 is the file handle.
* The Gauss file will be called "XYDATA."
* The column will be named by the strings in the character array xyname.
* ^xyname means use the names in the character string.
* 0, 8 gives double precision real numbers.
```

```
*****************************************************
*/
create f1 = xydata with ^xynames, 0, 8;
/*
*****************************************************
* Next read data into the Gauss file by using the writer command.
* f1 is the file handle defined in previous command.
* xydata is the data matrix that we just created.
* writer returns the number of rows read to f1.
* If it is not rows(xydata), something bad happended.
*****************************************************
*/
if writer(f1,xydata) /= rows(xydata);
errorlog "Conversion of XYDATA to Gauss File did not work";
endif;
closeall f1;


@ Plot Y versus X_i @
_plctrl  = -1; @ Plot symbols and no lines  @
for i0 (1,xdim,1); i = i0;
title(ynames $+ " versus " $+ xnames[i]);
xy(xdata[.,i],ydata); @ Plot y versus x_i  @
endfor;
graphset; @ Return to default graphs @
```

3.2 Bayesian Analysis

```
/*
****************************************
*   (C) Copyright 1999, Peter Lenk. All Rights Reserved.
* LINREG*.GSS
*   Does basic, linear regression model:
*
*   Y = X*beta + epsilon
*   epsilon is N(0,sigma^2I)
*   beta is N(b0,B0)
*   sigma^2 is IG(r0/2,s0/2)
*
*   library pgraph, plbam;
****************************************
*/
new;
inxy        = "xydata";         @ Name of Gauss file with X,Y data              @
outfile = "results1.dat";   @ Specify output file for saving results           @
                        @ outfile is a string variable that contains a file name @
/*
************************************
*   Initialize parameters for MCMC
************************************
*/
smcmc       = 1000;     @ number of iterations to save for analysis               @
skip        = 1;        @ Save every skip iterations                              @
nblow       = 500;      @ Initial transition iterations                           @
nmcmc       = nblow + skip*smcmc;   @ total number of iterations                  @

/*
************************************
*   Get data
************************************
*/
@ Input Gauss files @
open f1     = ^inxy;    @ Get Gauss file for X, Y data              @
               @ Opens Gauss file & assigns file handle f1 @
xydata      = readr(f1,rowsf(f1));
            @ readr reads in Gauss file with file handle f1.            @
            @ rowsf(f1) returns the number of rows in the Gauss file.   @
            @ readr reads rowsf(f1) rows, which is the entir dataset.   @
ci      = close(f1);
xynames     = setvars(inxy);            @ Get the variable names that accompnay X, Y data @
ynames      = xynames[rows(xynames)];
```

```
xnames      = "Constant"|xynames[1:rows(xynames)-1];

xdim        = cols(xydata)-1;           @ number of x variables                @
                                        @ cols(x) = # of columns of x          @
rankx       = xdim + 1;                 @ number of beta parameters            @
nobs        = rows(xydata);             @ number of observations               @
xdata       = ones(nobs,1)~xydata[.,1:xdim];
                                        @ design matrix, includes an intercept  @
                                        @ ~ sticks two matrices side by side    @
                                        @ ones(i,j) = i x j matrix of ones      @
                                        @ x[i,j] is the i,j element of x        @
                                        @ x[.,j] is the column j of  x          @
                                        @ x[.,j1:j2] are columns j1 to j2 of x  @
                                        @ x[i,.] is row i of x                  @
                                        @ x[i1:i2,.] are rows i1 to i2 of x     @
ydata       = xydata[.,cols(xydata)];

@ Sufficient statistics used in MCMC @
xtx         = xdata'xdata;
xty         = xdata'ydata;

@ Get MLE @
{bhat, bstd, sighat, rsquare} = regmle(xdata,ydata);

/*
************************************
*   Initialize Priors
************************************
*/

@ Beta is N(eb0, vb0) @
eb0     = zeros(rankx,1);       @ prior mean                              @
vb0     = 100*eye(rankx);       @ prior variance                          @
                            @eye(m) = m x m identity matrix               @
                            @ Generally, use big variance           @
vib0        = invpd(vb0);           @ invpd(x) = inverse of positive definite x     @
vieb0   = vib0*eb0;         @ used in full conditional of beta          @

@ Sigma2 is Inverted Gamma(r0/2, s0/2)          @
@ E(1/sigma2) = r0/s0 and Var(1/sigma2) = 2*r0/s0^2     @
@ Usually pick r0 and s0 small and positive         @
r0  = 2; s0 = 2;
rn  = r0 + nobs;            @ Posterior shape parameters @

/*
```

```
************************************
*   Initialize MCMC
************************************
*/

@ Initial parameters for MCMC   @
@ Could initialize to MLE.      @
beta    = zeros(rankx,1);       @ regression coefficients   @
sigma   = 1;                    @ error std             @
sigma2  = sigma*sigma;      @ error variance            @

@ Define matrics for saving MCMC iterations @
betag   = zeros(smcmc, rankx);
sigmag  = zeros(smcmc,1);

/*
************************************
*   Do MCMC
************************************
*/
@ Do the initial transition period @
for i1 (1,nblow,1); imcmc = i1;
    call getbetasigma;
endfor;

for i1 (1,smcmc,1); imcmc = i1;     @ Save smcmc iterations             @
    for i2 (1,skip,1); jmcmc = i2;      @ Save every skip iterations    @
        call getbetasigma;
    endfor;
    betag[imcmc,.]  = beta';        @ Save regression coefficients      @
    sigmag[imcmc,.]     = sigma;        @ Save error std                    @
endfor;


/*
*********************************************
*  Analyze Results
*********************************************
*/
@ Compute posterior means and std from MCMC iterations @
sigm    = meanc(sigmag);
sigstd  = stdc(sigmag);
bm      = meanc(betag);
bs      = stdc(betag);
yhat    = xdata*bm;
```

```
cy      = corrx(ydata~yhat);            @ Correlation between Y and Yhat @
cy2     = cy[1,2]*cy[1,2];
call outputanal;
@ Plot saved iterations against iterations number @
t   = seqa(nblow+skip,skip,smcmc);      @ saved iteration number @
title("Error STD versus Iteration");
xy(t,sigmag);
title("Coefficients versus Iteration");
xy(t,betag);


@ Compute the posterior density of sigma over a grid @
{sgrid,fs} = pdfsigma;

@ Compute the marginal posterior density of beta_j over a grid @
{bgrid, fb} = pdfbeta;
title("Posterior Density of the Error STD");
xy(sgrid,fs);


for fj (1,rankx,1); j = fj;
    title("Posterior Density of Coefficient for " $+ xnames[j]);
    xy(bgrid[.,j],fb[.,j]);
endfor;

graphset;           @ Return to default graphs @
end;

/*
************************************************
*   REGMLE
*   Compute MLE for simple regression
*   INPUT:
*       XDATA   = Design matrix
*       YDATA   = Dependent Variable
*   OUTPUT:
*       BHAT    = MLE for regression coefficients
*       BSTD    = STD Error of beta
*       Sighat  = Error STD
*       Rsqure  = R-Squared
************************************************
*/
PROC (4) = regmle(x,y);
local xtx, xtxi, xty, b, yhat, resid, sse, s, sst, r2, bstd;
    xtx     = x'x;
```

```
    xtxi        = invpd(xtx);               @ Inverse of xtx           @
    xty     = x'y;
    b       = xtxi*xty;             @ MLE of beta           @
    yhat        = x*b;                  @ Predicted y values        @
    resid   = y - yhat;             @ Residuals             @
    sse     = resid'resid;              @Sums of Squares Error  @
    s       = sqrt(sse/nobs);           @ Error STD             @
    sst     = y - meanc(y);
    sst     = sst'sst;                  @ SS Total              @
    r2      = 1 - sse/sst;              @ R-squared             @
    bstd        = s*sqrt(diag(xtxi));
retp(b,bstd, s,r2);                     @ Return to main program    @
endp;

/*
*********************************************
*  GETBETASIGMA
*   Generate one MCMC random deviate of beta and sigma
*   No input or output.
*   Data & values are passed through global variables
*********************************************
*/
PROC (0) = getbetasigma;
local  ebn, vibn, vbn, vbn12, sse, sn;
/*
*****************************************************
*  Generate beta:
*   beta is N(ebn, vbn);
*   vbn = (X'X/sigma2 + Vb0^{-1} )^{-1}     is posterior variance
*   ebn = vbn*(X'Y/sigma2 + Vb0^{-1}*eb0)   is posteriro mean
*****************************************************
*/
ebn     = xty/sigma2 + vieb0;       @ Posterior variance * ebn = posterior mean of beta     @
vibn    = xtx/sigma2 + vib0;        @ Inverse of posterior variance of beta             @
vbn     = invpd(vibn);
vbn12   = chol(vbn);
ebn     = vbn*ebn;
beta    = ebn + vbn12'rndn(rankx,1);

/*
*****************************************************
*  Generate sigma2
*   sigma2 is IG(rn/2, sn/2)
*   rn  = r0 + nobs
*   sn  = s0 + (y - X*beta)'(y - X*beta)
```

```
*****************************************************
*/
sse     = ydata - xdata*beta;
sse     = sse'sse;
sn      = s0 + sse;
sigma2  = sn/(2*rndgam(1,1,rn/2));
@ rndgam(i,j,alpha) generates a matrix of gamma random variables with   @
@ shape parameters alpha and scale parameter beta = 1               @
@ mean = alpha and variance = alpha                          @
sigma   = sqrt(sigma2);         @ Error STD                  @
endp;


/*
************************************************
*  OUTPUTANAL
*   Does analysis of output and save some results
************************************************
*/
PROC (0) = outputanal;

local  bout, sout, fmtn1, fmtn2, fmts1, fmts2;
format 10,5;                          @ Default format                          @
let fmtn1[1,3]  = "*.*lf" 10 5;       @ Format for printing numeric variable    @
let fmtn2[1,3]  = "*.*lf" 10  0;      @ Format for numeric variable, no decimal @

let fmts1[1,3]  = "-*.*s" 10 9;       @ Format for alpha, left justify          @
let fmts2[1,3]  = "*.*s" 10 9;        @ Format for alpha, right justify         @
output file = ^outfile reset;         @ outfile is the file handle for the output file  @
                                      @ Route printed output to the defined by outfile  @
print "Results from LINREG1A.GAS";
print "Bayesian analysis of linear regression model using MCMC.";
print "Ouput file: " getpath(outfile);          @ File assigned to file handle outfile @
datestr(date);                                  @ Print the current data              @
print;
print "Model";
print "Y = X*beta + epsilon";
print "Number of observations          = " nobs;
print "Number of independent variables = " xdim "(excluding the intercept).";
print "Summary Statistics";
call sumstats(xynames,xydata,fmts1,fmts2,fmtn1);    @ Print summary statistics @
print;
print;
print "-----------------------------------------";
print "MLE Analysis";
```

```
print;
print "Number of observations        = " nobs;
print "Number of dependent variables = " xdim;
print;
print "R-Squared     = " rsquare;
print "Multiple R    = " sqrt(rsquare);
print "MLE Error STD = " sighat;
print;
print "Estimated Regression Coefficents";
sout    = {"Variable"  "MLE"  "StdError"};
call outitle(sout,fmts1,fmts2);
bout    = xnames~bhat~bstd;
call outmat(bout,fmts1,fmtn1);

print "-----------------------------------------";
print;
print "MCMC Analysis";
print;
print "Total number of MCMC iterations                 = " nmcmc;
print "Number of iterations used in the analysis       = " smcmc;
print "Number in transition period                     = " nblow;
print "Number of iterations between saved iterations = " skip-1;
print;
print "Number of observations        = " nobs;
print "Number of dependent variables = " xdim " (excluding the intercept)";
@ Compute posterior means and std @
print;
print "Bayes R-Square   =   " cy2;
print "Bayes Multiple R =   " cy[1,2];
print;

print "Error Standard Deviation";
print "Posterior mean of sigma = " sigm;
print "Posterior STD  of sigma = " sigstd;
print;
print "Regression Coefficients";
sout    = {"Variable" "PostMean" "PostSTD"};
call outitle(sout,fmts1,fmts2);

bout    = xnames~bm~bs;
call outmat(bout,fmts1,fmtn1);
output off;
closeall;
endp;
```

```
/*
**********************************
* PDFSIGMA
*   Computes posterior density if sigma over a grid.
*   INPUT
*       Uses global variables
*   OUTPUT
*       sgrid   = grid for the density
*       fs      = posterior pdf on sgrid
**********************************
*/
PROC (2) = pdfsigma;
local smax, smin, ms, sdelta, sgrid, scon, fs, s2grid, lnsgrid, i0, i,
sse, sn;
@ compute posterior density of sigma over a grid @
smax    = maxc(sigmag);
        @ maxc(x) returns maximum of x          @
smin    = minc(sigmag);
        @ minc(x) returns minimum of x              @
@ Get grid for plotting     @
ms      = 100;
sdelta  = (smax-smin)/ms;
sgrid   = seqa(smin, sdelta, ms+1);
scon    = ln(2) - lnfact(1+rn/2);
fs      = zeros(ms+1,1);
s2grid  = sgrid.*sgrid;
lnsgrid = ln(sgrid);
for i0 (1,smcmc,1); i = i0;
    sse = ydata - xdata*(betag[i,.]');
    sse = sse'sse;
    sn  = s0 + sse;
    fs  = fs + exp(scon + rn*ln(sn/2)/2 + (0.5 - rn)*lnsgrid - sn/(2*s2grid));
endfor;
fs = fs/smcmc;
fs  = fs/intsim(fs,sdelta);     @ intsim is simpson's integration @
retp(sgrid,fs);
endp;

/*
**************************************
* PDFBETA
*   Computes marginal posterior density of beta over a grid.
*   These are marginal, univariate densities.
*   INPUT
*       Uses globals.
```

```
*   OUTPUT
*       bgrid   = each row corresponds to grid for beta_j
*       fb      = each row corresponds to marginal density for beta_j
*************************************
*/
PROC (2) = pdfbeta;
local mb, bgrid, bmax, bmin, bdelta, fj, j, fb, bcon, sigma, sigma2,
vibn, vbn, ebn, vbnd;
@ Plot posterior density for beta @
mb      = 100;
bgrid   = zeros(mb+1,rankx);
bmax    = bm + 5*bs;
bmin    = bm - 5*bs;
bdelta  = (bmax - bmin)/mb;
for fj (1,rankx,1) ; j = fj;
    bgrid[.,j] = seqa(bmin[j],bdelta[j], mb+1);
endfor;

fb      = zeros(mb+1,rankx);
bcon    = -ln(2*pi)/2;
for fj (1,smcmc, 1); j = fj;
    sigma   = sigmag[j];
    sigma2  = sigma^2;
    vibn    = (xtx/sigma2 + vib0);
    vbn     = invpd(vibn);                  @ Posterior variance given beta     @
    ebn     = vbn*(xty/sigma2 + vieb0);     @ Posterior mean given beta         @
    vbnd    = diag(vbn);                    @ diagonal elements                 @
    fb      = fb + exp(bcon - 0.5*ln(vbnd)' - ((bgrid - ebn')^2)./(2*vbnd') );
endfor;
fb = fb/smcmc;
fb = fb./(intsim(fb,bdelta)');

retp(bgrid, fb);
endp;



/*
************************************************
* OUTITLE
*   Prints header for columns of numbers.
*   INPUT
*       a   = character row vector of column names
*       fmts1   = format for first column
*       fmts2   = format for second column
```

```
*   OUTPUT
*       None
*******************************************
*/
PROC (0) = outitle(a,fmt1,fmt2);
local mask, fmt, flag, ncols;
ncols   = cols(a);
mask    = zeros(1,ncols);
fmt     = fmt1|(ones(ncols-1,1).*.fmt2);
flag    = printfm(a,mask,fmt);
print;
endp;
/*
***********************************************
* OUTMAT
*   Outputs a matrix:
*   (Character Vector)~(Numeric matrix);
*   The entries in the numeric matrix have the same format
*   INPUT
*       bout        = matrix to be printed
*       fmts        = format for string
*       fmtn        = format for numeric matrix
*   OUTPUT
*       None
*******************************************
*/
PROC (0) = outmat(bout,fmts,fmtn);
local fmt,mask,flag,ncols, nrows;
ncols       = cols(bout);
nrows       = rows(bout);
fmt         = fmts|(ones(ncols-1,1).*.fmtn);
mask        = zeros(nrows,1)~ones(nrows,ncols-1);
flag        = printfm(bout,mask,fmt);
print;
endp;

/*
***************************************************
* SUMSTATS
*   Prints summary statistics for a data matrix
*   INPUT
*       names       = charater vector of names
*       data        = data matrix
*       fmts1       = format for string
*       fmts2       = format for string
```

```
*       fmtn        = format for numbers
*   OUTPUT
*       None
*********************************************
*/
PROC (0) = sumstats(names,data,fmts1,fmts2,fmtn);
local a, bout;
a   = {"Variable" "Mean" "STD" "MIN" "MAX"};
call outitle(a,fmts1,fmts2);
bout    = names~meanc(data)~stdc(data)~minc(data)~maxc(data);
call outmat(bout,fmts1,fmtn);
endp;
```

3.3 Autocorrelated Errors

```
/*
****************************************
*   ARE1.GSS
*   linear regression model + AR Errors
*
*   Y = X*beta + epsilon
*   epsilon_t = rho*epsilon_{t-1} + zeta_t
*   zeta_t iid N(0,sigma^2);
*
*   Priors
*
*
*   beta is N(b0,B0)
*   sigma^2 is IG(r0/2,s0/2)
*   rho is uniform on [-1,1];
*
*   Issue the command:
*   library pgraph, plbam;
*   before running.
****************************************
*/
new;
inxy        = "xydata";         @ Name of Gauss file with X,Y data              @
outfile = "results1.dat";   @ Specify output file for saving results           @
                        @ outfile is a string variable that contains a file name @
/*
************************************
*   Initialize parameters for MCMC
************************************
*/
smcmc       = 1000;     @ number of iterations to save for analysis               @
skip        = 1;        @ Save every skip iterations                              @
nblow       = 1000;     @ Initial transition iterations                           @
nmcmc       = nblow + skip*smcmc;   @ total number of iterations                  @

/*
************************************
*   Get data
************************************
*/
xydata  = {
62      1   0   0   9.300964847     ,
62.25   0   1   0   9.31031088      ,
```

```
62.5        0   0   1   9.242491685     ,
62.75   0   0   0   9.361520061     ,
63      1   0   0   9.321018155     ,
63.25   0   1   0   9.369197279     ,
63.5        0   0   1   9.246424715     ,
63.75   0   0   0   9.40563659      ,
64      1   0   0   9.383025003     ,
64.25   0   1   0   9.426933948     ,
64.5        0   0   1   9.309481077     ,
64.75   0   0   0   9.405397557     ,
65      1   0   0   9.464221198     ,
65.25   0   1   0   9.489339612     ,
65.5        0   0   1   9.363198131     ,
65.75   0   0   0   9.50936384      ,
66      1   0   0   9.502659228     ,
66.25   0   1   0   9.522691945     ,
66.5        0   0   1   9.371714203     ,
66.75   0   0   0   9.528003453     ,
67      1   0   0   9.457957551     ,
67.25   0   1   0   9.50081259      ,
67.5        0   0   1   9.307902309     ,
67.75   0   0   0   9.388296608     ,
68      1   0   0   9.590930935     ,
68.25   0   1   0   9.569900363     ,
68.5        0   0   0   9.58463648      ,
69      1   0   0   9.554295277     ,
69.25   0   1   0   9.590206284     ,
69.5        0   0   1   9.502208538     ,
69.75   0   0   0   9.612942716     ,
70      1   0   0   9.530839779     ,
70.25   0   1   0   9.5922767       ,
70.5        0   0   1   9.53270342      ,
70.75   0   0   0   9.629857772     ,
71      1   0   0   9.587856805     ,
71.25   0   1   0   9.618832258     ,
71.5        0   0   1   9.584014436     ,
71.75   0   0   0   9.659631012     ,
72      1   0   0   9.685051414     ,
72.25   0   1   0   9.729132192     ,
72.5        0   0   1   9.644655112     ,
72.75   0   0   0   9.746657306     ,
73      1   0   0   9.786942924     ,
73.25   0   1   0   9.797592911     ,
73.5        0   0   1   9.697769625     ,
73.75   0   0   0   9.750624088     ,
```

```
74      1   0   0   9.73739145      ,
74.25   0   1   0   9.776250453     ,
74.5        0   0   1   9.779899324     ,
74.75   0   0   0   9.7895876       ,
75      1   0   0   9.707033295     ,
75.25   0   1   0   9.798042567     ,
75.5        0   0   1   9.784017688     ,
75.75   0   0   0   9.816413505     ,
76      1   0   0   9.871193346     ,
76.25   0   1   0   9.902312947     ,
76.5        0   0   1   9.792986685     ,
76.75   0   0   0   9.8580496       ,
77      1   0   0   9.964438819     ,
77.25   0   1   0   9.987259585     ,
77.5        0   0   1   9.943959343     ,
77.75   0   0   0   10.00549806     ,
78      1   0   0   9.9986647       ,
78.25   0   1   0   10.0753827      ,
78.5        0   0   1   9.985574693     ,
78.75   0   0   0   10.05099808     ,
79      1   0   0   10.09973943     ,
79.25   0   1   0   10.07600662     ,
79.5        0   0   1   9.955004308     ,
79.75   0   0   0   10.00015639     ,
80      1   0   0   9.982926704     ,
80.25   0   1   0   9.966892296     ,
80.5        0   0   1   9.903762747     ,
80.75   0   0   0   10.00828505     ,
81      1   0   0   9.969187596     ,
81.25   0   1   0   10.05141121     ,
81.5        0   0   1   9.937603098     ,
81.75   0   0   0   9.954898358     ,
82      1   0   0   9.950175009     ,
82.25   0   1   0   10.03861225     ,
82.5        0   0   1   9.906060277     ,
82.75   0   0   0   9.962198914     ,
83      1   0   0   10.00157806     ,
83.25   0   1   0   10.06937957     ,
83.5        0   0   1   10.01226763     ,
83.75   0   0   0   10.09340417     ,
84      1   0   0   10.11423726     ,
84.25   0   1   0   10.14908061     ,
84.5        0   0   1   10.07350585     ,
84.75   0   0   0   10.12767173     ,
85      1   0   0   10.12208808     ,
```

```
85.25   0   1   0   10.13992641     ,
85.5        0   0   1   10.06557971     ,
85.75   0   0   0   10.14911142     ,
86      1   0   0   10.16982431     ,
86.25   0   1   0   10.23810896     ,
86.5        0   0   1   10.15732984     ,
86.75   0   0   0   10.2111847      ,
87      1   0   0   10.25869966     ,
87.25   0   1   0   10.28994787     ,
87.5        0   0   1   10.18317873     ,
87.75   0   0   0   10.27318427     ,
88      1   0   0   10.3654338      ,
88.25   0   1   0   10.39715941     ,
88.5        0   0   1   10.30923406     ,
88.75   0   0   0   10.37861597     ,
89      1   0   0   10.41277977     ,
89.25   0   1   0   10.41312714     ,
89.5        0   0   1   10.30625996     ,
89.75   0   0   0   10.38282552     ,
90      1   0   0   10.37292315     ,
90.25   0   1   0   10.42929513     ,
90.5        0   0   1   10.36148037     ,
90.75   0   0   0   10.38365018     ,
91      1   0   0   10.32921477     ,
91.25   0   1   0   10.37742829     ,
91.5        0   0   1   10.32462605     ,
91.75   0   0   0   10.34206523     ,
92      1   0   0   10.39018415     ,
92.25   0   1   0   10.4288262      ,
92.5        0   0   1   10.36782724     ,
92.75   0   0   0   10.40494997     ,
93      1   0   0   10.42753479     ,
93.25   0   1   0   10.46862791     ,
93.5        0   0   1   10.38913063     ,
93.75   0   0   0   10.44468483     ,
94      1   0   0   10.48290215     ,
94.25   0   1   0   10.52855678     ,
94.5        0   0   1   10.48603355     ,
94.75   0   0   0   10.52689472     ,
95      1   0   0   10.54136704     ,
95.25   0   1   0   10.56097012     ,
95.5        0   0   1   10.49717854     ,
95.75   0   0   0   10.53841034     ,
96      1   0   0   10.55943978     ,
96.25   0   1   0   10.57906298     ,
```

```
96.5        0   0   1   10.53096768     ,
96.75   0   0   0   10.58920094     ,
97      1   0   0   10.55873256     ,
97.25   0   1   0   10.6049277      ,
97.5        0   0   1   10.55745908     ,
97.75   0   0   0   10.61346125     ,
98      1   0   0   10.56329119     ,
98.25   0   1   0   10.57158074     ,
98.5        0   0   1   10.51375015     ,
98.75   0   0   0   10.57867359     ,
99      1   0   0   10.57846729     ,
99.25   0   1   0   10.62615552     ,
99.5        0   0   1   10.49607128     ,
99.75   0   0   0   10.70802127     ,
100         1   0   0   10.55840854

};
xynames = { "Year", "Q1", "Q2", "Q3", "Sales"};

ynames      = xynames[rows(xynames)];



xdim        = cols(xydata)-1;           @ number of x variables                @
                                        @ cols(x) = # of columns of x          @
rankx       = xdim+1;                   @ number of beta parameters            @
xnames      = "Const"|xynames[1:xdim];

nobs        = rows(xydata);             @ number of observations               @
xmat        = xydata[.,1:xdim];
xdata       = ones(nobs,1)~xmat;
                                        @ design matrix, includes an intercept  @
                                        @ ~ sticks two matrices side by side    @
                                        @ ones(i,j) = i x j matrix of ones      @
                                        @ x[i,j] is the i,j element of x        @
                                        @ x[.,j] is the column j of  x          @
                                        @ x[.,j1:j2] are columns j1 to j2 of x  @
                                        @ x[i,.] is row i of x                  @
                                        @ x[i1:i2,.] are rows i1 to i2 of x     @
ydata       = xydata[.,cols(xydata)];

@ Sufficient statistics used in MCMC @
xtx         = xdata'xdata;
xty         = xdata'ydata;
```

```
@ Get MLE @
{bhat, bstd, sighat, rsquare} = regmle(xdata,ydata);

/*
*************************************
*   Initialize Priors
*************************************
*/

@ Beta is N(eb0, vb0) @
eb0     = zeros(rankx,1);       @ prior mean                              @

vb0     = 100000*eye(rankx);        @ prior variance                          @
                            @eye(m) = m x m identity matrix             @
                            @ Generally, use big variance           @
vib0        = invpd(vb0);           @ invpd(x) = inverse of positive definite x     @
vib0        = zeros(rankx,rankx);
vieb0   = vib0*eb0;         @ used in full conditional of beta          @


@ E(1/sigma2) = r0/s0 and Var(1/sigma2) = 2*r0/s0^2     @
@ Usually pick r0 and s0 small and positive         @
r0  = 0; s0 = 0;
rn  = r0 + nobs;            @ Posterior shape parameters: Add one for the epsilon_0 @

/*
*************************************
*   Initialize MCMC
*************************************
*/

@ Initial parameters for MCMC   @
@ Could initialize to MLE.      @
beta    = zeros(rankx,1);       @ regression coefficients   @
sigma   = 1;                    @ error std             @
sigma2  = sigma*sigma;      @ error variance            @

rho     = 0;



@ Define matrics for saving MCMC iterations @
betag   = zeros(smcmc, rankx);
sigmag  = zeros(smcmc,1);
```

```
rhog        = zeros(smcmc,1);


/*
************************************
*   Do MCMC
************************************
*/
@ Do the initial transition period @
icount = 0;
for i1 (1,nblow,1); imcmc = i1;
    call getall;
    icount = icount + 1;

endfor;

for i1 (1,smcmc,1); imcmc = i1;     @ Save smcmc iterations             @
    for i2 (1,skip,1); jmcmc = i2;      @ Save every skip iterations    @
        call getall;
        icount = icount + 1;
    endfor;
    betag[imcmc,.]  = beta';        @ Save regression coefficients      @
    sigmag[imcmc,.]     = sigma;        @ Save error std                    @
    rhog[imcmc,.]       = rho;

endfor;


/*
*******************************
*  Analyze Results
*******************************
*/
@ Compute posterior means and std from MCMC iterations @
sigm    = meanc(sigmag);
sigstd  = stdc(sigmag);
rhom    = meanc(rhog);
rhos    = stdc(rhog);
betam   = meanc(betag);
betas   = stdc(betag);
yhb     = xdata*betam;
cy      = corrx(ydata~yhb);         @ Correlation between Y and Yhat @
cy2     = cy[1,2]*cy[1,2];

yhat    = xdata*bhat;       @ MLE prediction @
```

```
rhat    = ydata[2:nobs] - yhat[2:nobs];     @ One step ahead forecast residuals mle: no AR @
resid = ydata - yhb;
ypred = yhb[2:nobs] + rho*resid[1:nobs-1];
rhbc    = ydata[2:nobs] - ypred;               @ One step ahead forecast residuals Bayes @


PrmseML = sqrt(rhat'rhat/(nobs-1));
PrmseB   = sqrt(resid'resid/(nobs-1));
PrmseBC   = sqrt(rhbc'rhbc/(nobs-1));

call outputanal;
@ Plot saved iterations against iterations number @
t   = seqa(nblow+skip,skip,smcmc);      @ saved iteration number @


@_plctrl = -1;@
_plwidth = 8;
title("Ford's Quarterly Revenue vs Year");
xlabel("Year");
ylabel("Log Revenue");
_plegctl = { 1 7 62 10 };
/*************
             [1]  X,Y coordinate units: 1=plot coord, 2=inches, 3=Tek pixels
             [2]  Legend text font size. 1 <= size <= 9.
             [3]  X coordinate of lower left corner of legend box
             [4]  Y coordinate of lower left corner of legend box
*************/

_plegstr = "Observed\000Bayes AR\000ML IID";
xy(xdata[2:nobs,2],ydata[2:nobs]~ypred~yhat[2:nobs]);

title("Forecast Residuals vs Year");
ylabel("Residuals");
xy(xdata[2:nobs,2], rhbc~rhat);

graphset;
title("Posterior Distribution of rho");

xlabel("rho");
{f,m,p} = hist(rhog,25);
graphset;
```

```
end;

/*
*******************************
*   REGMLE
*   Compute MLE for simple regression
*   INPUT:
*       XDATA   = Design matrix
*       YDATA   = Dependent Variable
*   OUTPUT:
*       BHAT    = MLE for regression coefficients
*       BSTD    = STD Error of beta
*       Sighat  = Error STD
*       Rsqure  = R-Squared
*******************************
*/
PROC (4) = regmle(x,y);
local xtx, xtxi, xty, b, yhat, resid, sse, s, sst, r2, bstd;
    xtx     = x'x;
    xtxi    = invpd(xtx);               @ Inverse of xtx            @
    xty     = x'y;
    b       = xtxi*xty;             @ MLE of beta           @
    yhat        = x*b;                  @ Predicted y values        @
    resid   = y - yhat;             @ Residuals             @
    sse     = resid'resid;              @Sums of Squares Error  @
    s       = sqrt(sse/nobs);           @ Error STD             @
    sst     = y - meanc(y);
    sst     = sst'sst;                  @ SS Total              @
    r2      = 1 - sse/sst;              @ R-squared             @
    bstd        = s*sqrt(diag(xtxi));
retp(b,bstd, s,r2);                     @ Return to main program    @
endp;

/*
*********************************************
*  GETALL
*   Generate one MCMC random deviate of beta and sigma
*   No input or output.
*   Data & values are passed through global variables
*********************************************
*/
PROC (0) = getall;
```

```
local s1mr, ystar, xstar, vibn, vibn12, ebn,
resid, rstar, sn, v2, rhotop, rhobot, arho, brho, rv, rm;



s1mr = sqrt(1 - rho^2);

/*
************************************
*  Generate beta:
*   Do adjustment for AR Errors.
*   beta is N(ebn, vbn);
*   vbn = (X'X/sigma2 + Vb0^{-1} )^{-1}     is posterior variance
*   ebn = vbn*(X'Y/sigma2 + Vb0^{-1}*eb0)   is posteriro mean
************************************
*/

ystar       = ydata - rho*(0|ydata[1:nobs-1]);
xstar       = xdata - rho*( zeros(1,rankx) | xdata[1:nobs-1,.]);
xstar[1,.]      = s1mr*xstar[1,.];
ystar[1]        = s1mr*ystar[1];
vibn            =  xstar'xstar/sigma2 + vib0;
vibn12      = chol(vibn);                       @ vibn12'vibn12 = vibn                          @
ebn         = xstar'ystar/sigma2 + vieb0;
beta            = cholsol(ebn+vibn12'rndn(rankx,1), vibn12);
        @ cholsol is efficient method of solving linear equations if you have chol decompostion
        @ Suppose C'C = A                                                                   @
        @ Need to find x to solve A*x = b                                                   @
        @ x = cholsol(b, C) does it                                                         @


/*
***************************
*  Generate sigma2
*   sigma2 is IG(rn/2, sn/2)
*   rn  = r0 + nobs
*   sn  = s0 + (y - X*beta)'(y - X*beta)
***************************
*/
resid       = ydata - xdata*beta;
rstar           = resid - rho*(0|resid[1:nobs-1]);
rstar[1]        = rstar[1]*s1mr;

sn          = s0 + rstar'rstar;
sigma2      = sn/(2*rndgam(1,1,rn/2));
```

```
sigma       = sqrt(sigma2);         @ Error STD                 @



/*
***************************
* Generate rho
*   Use Damien's method to handle the determinant factor.
*   Truncated normal on -1 to 1
*
* (See program TEST2.GSS to test the following identities.)
* Smat = sigma2*cor/(1-rho^2)  where
* cor  is Topelitz with (i,j) element rho^{|i-j|}.
* Det|Smat|^{-1/2} = sqrt((1 - rho^2))/(sigma2^{n/2})

***************************
*/

/*
***************************
* To include the determinate:
* V < sqrt(1 - rho^2)
* - sqrt( 1 - V^2 ) < rho < sqrt( 1 + V^2)
***************************
*/


v2      = (1 - rho^2)*rndu(1,1);
rhotop  = sqrt(1 - v2);
rhobot  = - rhotop;

@ Keep rho between -1 and 1 @
rhobot  = maxc(rhobot|-1);
rhotop  = minc(rhotop|1);


arho    = resid[2:nobs-1]'resid[2:nobs-1];
brho    = resid[2:nobs]'resid[1:nobs-1];
rv      = sigma2/arho;
rm      = brho/arho;
rho     = rndnab(1,1,rm,rv,rhobot,rhotop);      @ truncated normal from -1 to 1 @
```

```
endp;


/*
***********************************
*  OUTPUTANAL
*   Does analysis of output and save some results
***********************************
*/
PROC (0) = outputanal;

local  bout, sout, fmtn1, fmtn2, fmts1, fmts2;
format 10,5;                            @ Default format                          @
let fmtn1[1,3]  = "*.*lf" 10 5;         @ Format for printing numeric variable    @
let fmtn2[1,3]  = "*.*lf" 10  0;        @ Format for numeric variable, no decimal @

let fmts1[1,3]  = "-*.*s" 10 9;         @ Format for alpha, left justify          @
let fmts2[1,3]  = "*.*s" 10 9;          @ Format for alpha, right justify         @
output file = ^outfile reset;           @ outfile is the file handle for the output file  @
                                        @ Route printed output to the defined by outfile  @
print "Results from ARE1.GSS";
print "Bayesian analysis of linear regression model using MCMC.";
print "AR Error Terms.";
print "Ouput file: " getpath(outfile);          @ File assigned to file handle outfile @
datestr(date);                                  @ Print the current data               @
print;
print "Model";
print "Y = X*beta + epsilon";
print "epsilon_t = rho*epsilon_{t-1} + z_t";
print "Var(Z_t) = sigma^2";
print;
print "Number of observations          = " nobs;
print "Number of independent variables = " xdim "(excluding the intercept).";
print "Summary Statistics";
call sumstats(xynames,xydata,fmts1,fmts2,fmtn1);    @ Print summary statistics @
print;
print;
print "-----------------------------------------";
print "MLE Analysis";
print;
print "Number of observations        = " nobs;
print "Number of dependent variables = " xdim;
print;
```

```
print "R-Squared     = " rsquare;
print "Multiple R    = " sqrt(rsquare);
print "One-Step Ahead Predictive RMSE = " PrmseML;
print;
print "MLE Error STD = " sighat;
print;
print "Estimated Regression Coefficents";
sout    = {"Variable"  "MLE"  "StdError"};
call outitle(sout,fmts1,fmts2);
bout    = xnames~bhat~bstd;
call outmat(bout,fmts1,fmtn1);

print "-----------------------------------------";
print;
print "MCMC Analysis";
print;
print "Total number of MCMC iterations                 = " nmcmc;
print "Number of iterations used in the analysis       = " smcmc;
print "Number in transition period                     = " nblow;
print "Number of iterations between saved iterations = " skip-1;
print;
print "Number of observations        = " nobs;
print "Number of dependent variables = " xdim " (excluding the intercept)";
@ Compute posterior means and std @
print;
print "Bayes R-Square   =   " cy2;
print "Bayes Multiple R =   " cy[1,2];
print "One-Step Ahead Predictive RMS without AR Correction   = " PrmseB;
print "One-Step Ahead Predictive RMSE Corrected for AR Errors = " PrmseBC;
print;

print "Error Standard Deviation";
print "Posterior mean of sigma = " sigm;
print "Posterior STD  of sigma = " sigstd;
print;
print "Error Correlation";
print "Posterior mean of rho = " rhom;
print "Posterior STD  of rho = " rhos;
print;
print "Regression Coefficients";
sout    = {"Variable" "PostMean" "PostSTD"};
call outitle(sout,fmts1,fmts2);

bout    = xnames~betam~betas;
call outmat(bout,fmts1,fmtn1);
```

```
output off;
closeall;
endp;



/*
************************************************
* OUTITLE
*   Prints header for columns of numbers.
*   INPUT
*       a   = character row vector of column names
*       fmts1   = format for first column
*       fmts2   = format for second column
*   OUTPUT
*       None
*****************************************
*/
PROC (0) = outitle(a,fmt1,fmt2);
local mask, fmt, flag, ncols;
ncols   = cols(a);
mask    = zeros(1,ncols);
fmt     = fmt1|(ones(ncols-1,1).*.fmt2);
flag    = printfm(a,mask,fmt);
print;
endp;
/*
*********************************************
* OUTMAT
*   Outputs a matrix:
*   (Character Vector)~(Numeric matrix);
*   The entries in the numeric matrix have the same format
*   INPUT
*       bout        = matrix to be printed
*       fmts        = format for string
*       fmtn        = format for numeric matrix
*   OUTPUT
*       None
*****************************************
*/
PROC (0) = outmat(bout,fmts,fmtn);
local fmt,mask,flag,ncols, nrows;
ncols       = cols(bout);
nrows       = rows(bout);
```

```
fmt         = fmts|(ones(ncols-1,1).*.fmtn);
mask        = zeros(nrows,1)~ones(nrows,ncols-1);
flag        = printfm(bout,mask,fmt);
print;
endp;

/*
************************************************
* SUMSTATS
*   Prints summary statistics for a data matrix
*   INPUT
*       names       = charater vector of names
*       data        = data matrix
*       fmts1       = format for string
*       fmts2       = format for string
*       fmtn        = format for numbers
*   OUTPUT
*       None
*********************************************
*/
PROC (0) = sumstats(names,data,fmts1,fmts2,fmtn);
local a, bout;
a   = {"Variable" "Mean" "STD" "MIN" "MAX"};
call outitle(a,fmts1,fmts2);
bout    = names~meanc(data)~stdc(data)~minc(data)~maxc(data);
call outmat(bout,fmts1,fmtn);
endp;
```

Chapter 4

Multivariate Regression

4.1 Data Generation

```
/*
***********************************************
*  GMULREG.GSS
* Genenerates data for multiple regression analysis:
*
* Y = X*B + U
*
* Y is a nobs by mvar matrix
* Rows of Y correspond to subjects.
* Columns of Y correspond to variables.
*
* X is a nobs by rankx design matrix
* Rows of X correspond to subjects.
* Columns of X corresponds to variables.
* B is a rankx by mvar matrix of regression coefficients.
* U is matrix normal.
* The mean of U is zero.
* The rows of U are independent of each other.
* Each row of U has the same covariance matrix Sigma.
*
* Y has a matrix normal distribution.
* Write Y as row vectors:
* Y = { Y_1', ..., Y_n'}
* so that Y_i is a mvar vector of observations for subject i.
*
* Define yv = vec(Y') = {Y_1, Y_2, ..., Y_n}
* yv is a nobs*mvar vector that stacks the observations for
* each individual.
*
* Then yv = vec(B'*X') + vec(U').
*
* One can verify that
* vec(B'*X') = (X.*.I)*vec(B')
* where .*. is Kroneck product and
* I is a mvar by mvar identity matrix.
* Define beta = vec(B'), which is a rankx*mvar vector.
*
* Put epsilon = vec(U').
* Then epsilon is a nobs*mvar vector.
* It has a normal distriubtion with mean 0 and covariance matrix
* Var(epsilon) = I.*.Sigma
* where I is a nobs by nobs identity matrix.
*
```

```
* Then, we can write
* yv = (X.*.I)*beta + epsilon.
*
* It has density:
* (2*pi)^{nobs*mvar/2)|Sigma|^{-nobs/2}
* *exp( -0.5*(yv - (X.*.I)*beta)'(I.*.Sigma^{-1})(yv - (X.*.I)*beta) )
*
* Another way to write this is:
*
* (2*pi)^{nobs*mvar/2)|Sigma|^{-nobs/2}
* exp(-0.5*tr( Sigma^{-1}(Y - X*B)'(Y-X*B) ) )
* where "tr" is the trace of a matrix (sum of the diagonal elements).
*******************************************************
*/
new;
flagplot  = 1; @ 1 -> do a bunch of plots @
nobs = 100; @ Number of subjects  @
mvar =  4; @ Y_{ij} is mvar vector. Eg: mvar brands   @
rankx = 3; @ Rank of the design matrix @
xdim = rankx - 1; @ Number of X variables excludin the intercept @
pardim  = nobs|mvar|rankx;
@ Define some variable names for Gauss file @
@ Use string arrays @
a = seqa(1,1,mvar);
ynames = 0 $+ "Y" $+ ftocv(a,1,0);
@ ftocv converts a number to the corresponding character @
a = seqa(1,1,xdim);
xnames = 0 $+ "X" $+ ftocv(a,1,0);
xynames = xnames|ynames;

@ Generate X data. @
xdata = rndn(nobs,xdim); @ xdata does not have an intercept @
xmat = ones(nobs,1)~xdata;

@ Define true error variance Sigma @

@ First do chol decomposition (Gauss does upper triangular) @
sig12 = {
3 -2 1 4,
0 .5 3 -2,
0 0 2 -4,
0 0 0 1
};
sigmat = sig12'sig12;
```

```
@ Define true bmatt @
bmatt = {
1 2 -1 4,
-3 0 4 2,
2 .5 -4 6
};

@ Generate ydata @
ydata = xmat*bmatt + rndn(nobs,mvar)*sig12;
@ Wasn't that simple! @

xydata = xdata~ydata;

@ Create a Gauss file.  f1 is the file handle.  @
create f1 = xydata with ^xynames, 0, 8;
if writer(f1,xydata) /= rows(xydata);
errorlog "Conversion of XYDATA to Gauss File did not work";
endif;
closeall f1;
save pardim = pardim;

save sigmat = sigmat;
save bmatt = bmatt;

if flagplot == 1;
_plctrl = -1; @ plot with symbols only @
for i0 (1,mvar,1); i = i0;
for fj (1,xdim,1); j = fj;
title(" " $+ ynames[i] $+ " versus " $+ xnames[j]);
xy(xdata[.,j],ydata[.,i]);
endfor;
endfor;
graphset;
endif;


end;
```

4.2 Bayesian Analysis

```
/*
**************************************************
* MULREG.GSS
* Analyzes data form multivariate regression model:
*
* Y = X*B + U
*
* Y is a nobs by mvar matrix
* Rows of Y correspond to subjects.
* Columns of Y correspond to variables.
*
* X is a nobs by rankx design matrix
* Rows of X correspond to subjects.
* Columns of X corresponds to variables.
*
* B is a rankx by mvar matrix of regression coefficients.
*
* U is matrix normal.
* The mean of U is zero.
* The rows of U are independent of each other.
* Each row of U has the same covariance matrix Sigma.
*
* Prior:
* vec(B') is N(u0,v0);
* Sigma is Inverted Wishart(f0,g0)
* f0 is prior df.
* g0 is prior scale matrix.
*
* NOTES:
* Y has a matrix normal distribution.
* Write Y as row vectors:
* Y = { Y_1', ..., Y_n'}
* so that Y_i is a mvar vector of observations for subject i.
*
* Define yv = vec(Y') = {Y_1, Y_2, ..., Y_n}
* yv is a nobs*mvar vector that stacks the observations for
* each individual.
*
* Then yv = vec(B'*X') + vec(U').
*
* One can verify that
* vec(B'*X') = (X.*.I)*vec(B')
* where .*. is Kroneck product and
```

```
* I is a mvar by mvar identity matrix.
* Define beta = vec(B'), which is a rankx*mvar vector.
*
* Put epsilon = vec(U').
* Then epsilon is a nobs*mvar vector.
* It has a normal distriubtion with mean 0 and covariance matrix
* Var(epsilon) = I.*.Sigma
* where I is a nobs by nobs identity matrix.
*
* Then, we can write
* yv = (X.*.I)*beta + epsilon.
*
* It has density:
* (2*pi)^{nobs*mvar/2)|Sigma|^{-nobs/2}
* *exp( -0.5*(yv - (X.*.I)*beta)'(I.*.Sigma^{-1})(yv - (X.*.I)*beta) )
*
* Another way to write this is:
*
* (2*pi)^{nobs*mvar/2)|Sigma|^{-nobs/2}
* exp(-0.5*tr( Sigma^{-1}(Y - X*B)'(Y-X*B) ) )
* where "tr" is the trace of a matrix (sum of the diagonal elements).
*
* MATRIX Fun Fact
* vec(B*C) = (I.*.B)*vec(C)
* = (C'.*.I)*vec(B)
* = (C'.*.B)*vec(I)
*******************************************************
*/

new;
outfile = "results1.dat"; @ Specify output file for saving results  @
@ outfile is a string variable that contains a file name  @
inxy  = "xydata"; @ Name of Gauss file with X,Y data @
flagtrue = 1; @ 1 -> knows true parameters from simulation @

/*
***********************************************
* Initialize parameters for MCMC
***********************************************
*/
smcmc = 500; @ number of iterations to save for analysis  @
skip = 1; @ Save every skip iterations @
nblow = 500; @ Initial transition iterations  @
nmcmc = nblow + skip*smcmc; @ total number of iterations @
```

```
/*
************************************************
* Get data
************************************************
*/
load pardim = pardim; @ pardim give nobs, mvar, rankx @
nobs = pardim[1];
mvar = pardim[2];
rankx = pardim[3]; @ rank of the design matrix  @
xdim = rankx - 1; @ does not include intercept  @
@ Input Gauss files @
open f1  = ^inxy; @ Get Gauss file for X, Y data  @
@ Opens Gauss file & assigns file handle f1 @
xydata = readr(f1,rowsf(f1));
@ readr reads in Gauss file with file handle f1. @
@ rowsf(f1) returns the number of rows in the Gauss file.  @
@ readr reads rowsf(f1) rows, which is the entir dataset. @
ci = close(f1);
if not nobs == rows(xydata);
errorlog "Rows of data matrix are wrong: Check pardim.";
endif;
@ First xdim columns of xydata are the independent variables  @
@ Columns xdim+1 to xdim+mvar are the dependent variables @
if not xdim+mvar == cols(xydata);
errorlog "Columns of data matrix are wrong: Check pardim.";
endif;
xdata = ones(nobs,1)~xydata[.,1:xdim];
ydata = xydata[.,xdim+1:xdim+mvar];
yv = vecr(ydata);
@ Get the variable names that accompnay X, Y data @
xynames = setvars(inxy);
xnames = xynames[1:xdim];
xnames = "Constant"|xnames;
ynames = xynames[xdim+1:xdim+mvar];


/*
************************************************
* Compute MLE
************************************************
*/
xtx = xdata'xdata;
xtxi = invpd(xtx);
xty = xdata'ydata;
bhat = xtxi*xty; @ bhat = (X'X)^{-1}X'Y @
```

```
resid = ydata - xdata*bhat;
sighat = resid'resid/nobs;
@ Define betah = vec(bhat') @
@ The betah = [ (X'X)^{-1}X' .*. I]* vec(Y') @
@ Var(betah) = (X'X)^{-1} .*. Sigma         @
bhatvar = xtxi .*. sighat;
bstderr = sqrt( reshape( diag(bhatvar),rankx,mvar) );

/*
************************************************
* Initialize Priors
************************************************
*/


@ Prior bmat is N(u0,v0) @
@ put beta = vec(bmat') @
bdim = rankx*mvar;
u0  = zeros(bdim,1);
v0    = 100*eye(bdim); @ thdim = rankx*nparz    @
v0i   = invpd(v0);     @ used in updating theta @
v0iu0  = v0i*u0;         @ used in updating theta @

@ Prior for sigma is IW(f0, g0) @
f0  = mvar+2; f0n = f0 + nobs;
g0i = eye(mvar);

/*
************************************************
* Initialize MCMC
************************************************
*/
bmat = zeros(rankx,mvar);
beta  = vecr(bmat);
sigma = eye(mvar);
sigmai = invpd(sigma);

@ Define data structures for saving iterates & computing posterior means & std @
betag = zeros(smcmc,bdim);
c = mvar*(mvar+1)/2;
sigmag = zeros(smcmc,c); @ save iterations for sigma  @


/*
************************************************
```

```
* Do MCMC
************************************************
*/
@ Do the initial transition period @
for i1 (1,nblow,1); imcmc = i1;
{bmat, sigma, sigmai} = getmulreg(ydata,xdata,xtx,bmat,sigma,sigmai,v0i,v0iu0,f0n,g0i);
endfor;

for i1 (1,smcmc,1); imcmc = i1; @ Save smcmc iterations  @
for i2 (1,skip,1); jmcmc = i2; @ Save every skip iterations  @
{bmat, sigma, sigmai} = getmulreg(ydata,xdata,xtx,bmat,sigma,sigmai,v0i,v0iu0,f0n,g0i);
endfor;
@ Save the random deviate. @
betag[imcmc,.] = vecr(bmat)';
sigmag[imcmc,.] = vech(sigma)';
@ vech({1 2 3, 4 5 6, 7 8 9}) = {1, 4 5, 7 8 9} @
@ xpnd is the inverse operator of vech @
endfor;


/*
***********************************************
* Compute Posterior Means and STD
***********************************************
*/
betam = meanc(betag);
bmatm = reshape(betam,rankx,mvar);
sigmam = xpnd(meanc(sigmag)); @ xpnd reconstructs symmetric matrix @

betas = stdc(betag);
bmats = reshape(betas,rankx,mvar);
sigmas = xpnd(stdc(sigmag));

@ Compute predictive value of Y_{ij} @
yhat = xdata*bmatm;
resid = ydata - yhat;
stderr = sqrt(diag( resid'resid/nobs) );
@ Pick out each dimension of Y_{ij} and compute fit statistics @
multir = zeros(mvar,1);
rsquare = zeros(mvar,1);
for fm (1,mvar,1); m = fm;
cm = corrx(ydata[.,m]~yhat[.,m]);
multir[m] = cm[1,2];
rsquare[m] = cm[1,2]^2;
endfor;
```

```
/*
*********************************************
* Do some output
*********************************************
*/
call outputanal;


@ Plot saved iterations against iterations number @
t  = seqa(nblow+skip,skip,smcmc); @ saved iteration number @
title("beta = vec(B') versus Iteration");
xy(t,betag);
title("Error Variance versus Iteration");
xy(t,sigmag);
graphset;

end;

/*
*********************************************
* GETMULREG
* Generate multivariate regression parameters.
* Ydata = Xdata*bmat + epsilon
*
* INPUT
* ydata = dependent variables
* xdata = independet variables
* xtx = xdata'xdata
*
* bmat = current value of coefficient matrix
* sigma = current value of covariance matrix
* sigmai = its inverse
* v0i = prior precisions for bmat
* v0iu0 = prior precision*prior mean for bmat
* f0n = posterior df for sigma
* g0i = prior scaling matrix inverse for sigma

*
* OUTPUT
* bmat = updated rankx x mvar coefficient matrix
* sigma = updated variance
* sigmai = updated inverse of sigma
*
```

```
* Calling Statement:
{bmat, sigma, sigmai} = getmulreg(ydata,xdata,xtx,bmat,sigma,sigmai,v0i,v0iu0,f0n,g0i);
*****************************************
*/
PROC (3) = getmulreg(ydata,xdata,xtx,bmat,sigma,sigmai,v0i,v0iu0,f0n,g0i);
local vb12, ubn, beta, bdim, resid, gni, gn, rankx, mvar;

rankx = rows(bmat);
mvar = cols(ydata);
bdim = rankx*mvar;

/*
*****************************************
* Generate bmat from N_{rankx x mvar}(M,v)
* beta = vec(bmat')
* beta is N(u,V) whee u = vec(M');
* V = (X'X.*.Sigma^{-1} + V_0^{-1})^{-1}
* u = V*( (X'.*.Sigma^{-1})*vec(Y') + V_0^{-1}u_0 )
*****************************************
*/

vb12 = chol(xtx.*.sigmai + v0i);
ubn = ( (xdata').*.sigmai )*vecr(ydata) + v0iu0;
beta = cholsol(ubn + vb12'rndn(bdim,1), vb12);
bmat = reshape(beta,rankx,mvar);

/*
***************************************
* Generate Sigma
*  Sigma^{-1} is Wishart df = f0n, scale matrix = gn
***************************************
*/
resid  = ydata - xdata*bmat;
gni = g0i + resid'resid;
gn   = invpd(gni);
{sigmai, sigma}  = wishart(mvar,f0n,gn);

retp(bmat,sigma,sigmai);
endp;

/*
*************************************************************************
*  OUTPUTANAL
* Does analysis of output and save some results
*************************************************************************
```

```
*/
PROC (0) = outputanal;
format 10,5;
local  bout, sout, ebeta, sbeta, cb, rmse, fmtn1, fmtn2, fmts1, fmts2, a,b,
bmatt, sigmat;



if flagtrue == 1; @ Did a simulation. Get true parameters @
load bmatt  = bmatt;
load sigmat = sigmat;
endif;

let fmtn1[1,3] = "*.*lf" 10 5; @ Format for printing numeric variable  @
let fmtn2[1,3] = "*.*lf" 10  0; @ Format for numeric variable, no decimal @

let fmts1[1,3] = "-*.*s" 10 9; @ Format for alpha, left justify   @
let fmts2[1,3]  = "*.*s" 10 9; @ Format for alpha, right justify    @
format 10,5; @ Default pring format @


output file = ^outfile reset; @ outfile is the file handle for the output file  @
@ Route printed output to the defined by outfile  @
print "Results from MULReg1.GSS";
print "Bayesian Multivariate Regression using MCMC.";
print "Y = X*B + U";
print "U is matrix normal, mean 0, Var(U) = I .*. Sigma";
print;
print "Ouput file: " getpath(outfile); @ File assigned to file handle outfile @
datestr(date); @ Print the current data @
print;
print;
print "-----------------------------------------------------------";
print;
print "MCMC Analysis";
print;
print "Total number of MCMC iterations               = " nmcmc;
print "Number of iterations used in the analysis     = " smcmc;
print "Number in transition period                   = " nblow;
print "Number of iterations between saved iterations = " skip-1;
print;
print "-----------------------------------------------------------";
print "Number of observations          = " nobs;
print "Number of dependent variables   = " mvar;
print "Number of independnet variables = " rankx " (including intercept)";
```

```
print;
print;
print "Y = X*B + U";
print;
print " Summary Statistics for Y";
call sumstats(ynames,ydata,fmts1,fmts2,fmtn1);
print;
print " Summary Statistics for X";
call sumstats(xnames[2:rankx],xdata[.,2:rankx],fmts1,fmts2,fmtn1);
print;
print;
print "-----------------------------------------------------------------";
print "Statistics of Fit Measures for each Dimension";
sout = {"Variable" "Multi-R" "R-Sqr" "ErrorSTD"};
call outitle(sout,fmts1,fmts2);

bout = ynames~multir~rsquare~stderr;
call outmat(bout,fmts1,fmtn1);
print;
print "-----------------------------------------------------------------";
print;
print "Estimated Regression Coefficients";
sout = "  "~(ynames');
if flagtrue == 1;
print "True B";
call outitle(sout,fmts1,fmts2);
bout = xnames~bmatt;
call outmat(bout,fmts1,fmtn1);
print;
endif;
print "ML Estimates of B";
call outitle(sout,fmts1,fmts2);
bout = xnames~bhat;
call outmat(bout,fmts1,fmtn1);
print;
print "STD Error of MLE of B";
call outitle(sout,fmts1,fmts2);
bout = xnames~bstderr;
call outmat(bout,fmts1,fmtn1);
print;
if flagtrue == 1;
print "True B";
call outitle(sout,fmts1,fmts2);
bout = xnames~bmatt;
call outmat(bout,fmts1,fmtn1);
```

```
print;
endif;
print "Bayes Estimates of B";
print "Posterior Mean of B";
print outitle(sout,fmts1,fmts2);
bout = xnames~bmatm;
call outmat(bout,fmts1,fmtn1);
print;
print "Posterior STD of B";
call outitle(sout,fmts1,fmts2);
bout = xnames~bmats;
call outmat(bout,fmts1,fmtn1);
print;

print;
print "-------------------------------------------------------------";

print "Estimation of the error covariance Sigam";
sout = "   "~(ynames');

if flagtrue == 1;
print "True Sigma";
call outitle(sout,fmts1,fmts2);
bout = ynames~sigmat;
call outmat(bout,fmts1,fmtn1);
print;
endif;
print "ML Estimate of Sigma";
call outitle(sout,fmts1,fmts2);
bout = ynames~sighat;
call outmat(bout,fmts1,fmtn1);
print;
print "Bayes Estimates of Sigma";
print "Posterior Mean of Sigma";
call outitle(sout,fmts1,fmts2);
bout = ynames~sigmam;
call outmat(bout,fmts1,fmtn1);
print;
print "Posterior STD  of Sigma";
call outitle(sout,fmts1,fmts2);
bout = ynames~sigmas;
call outmat(bout,fmts1,fmtn1);
print;
print "=============================================================";
```

```
output off;
closeall;
endp;



/*
****************************************************************
* OUTITLE
* Prints header for columns of numbers.
* INPUT
* a  = character row vector of column names
* fmts1 = format for first column
* fmts2 = format for second column
* OUTPUT
* None
****************************************************************
*/
PROC (0) = outitle(a,fmt1,fmt2);
local mask, fmt, flag, ncols;
ncols = cols(a);
mask = zeros(1,ncols);
fmt = fmt1|(ones(ncols-1,1).*.fmt2);
flag = printfm(a,mask,fmt);
print;
endp;
/*
****************************************************************
* OUTMAT
* Outputs a matrix:
* (Character Vector)~(Numeric matrix);
* The entries in the numeric matrix have the same format
* INPUT
* bout = matrix to be printed
* fmts = format for string
* fmtn = format for numeric matrix
* OUTPUT
* None
****************************************************************
*/
PROC (0) = outmat(bout,fmts,fmtn);
local fmt,mask,flag,ncols, nrows;
ncols = cols(bout);
nrows = rows(bout);
```

```
fmt = fmts|(ones(ncols-1,1).*.fmtn);
mask = zeros(nrows,1)~ones(nrows,ncols-1);
flag = printfm(bout,mask,fmt);
print;
endp;

/*
************************************************************
* SUMSTATS
* Prints summary statistics for a data matrix
* INPUT
* names = charater vector of names
* data = data matrix
* fmts1 = format for string
* fmts2 = format for string
* fmtn = format for numbers
* OUTPUT
* None
************************************************************
*/
PROC (0) = sumstats(names,data,fmts1,fmts2,fmtn);
local a, bout;
a = {"Variable" "Mean" "STD" "MIN" "MAX"};
call outitle(a,fmts1,fmts2);
bout = names~meanc(data)~stdc(data)~minc(data)~maxc(data);
call outmat(bout,fmts1,fmtn);
endp;
```

Chapter 5

HB Regression: Interaction Model

5.1 Data Generation

```
/*
**********************************************
*  GHBREG2.GSS
* Generats data for HB Regression Model
* Y_i  = X_i*beta_i + epsilon_i
* beta_i = Theta'Z_i + delta_i
* B = Z*Theta + D
* epsilon_i is N(0,sigma2*I)
* delta_i is N(0,Lambda)
*--->Different Design Matrices
**********************************************
*/
new;
nobs = 100; @ Number of subjects  @
yrows = 5 + floor(10*rndu(nobs,1)); @ Number of observations per subject @
ntot = sumc(yrows);
sigmat = 5; @ True error STD @

lbd12 = {
5 .5 -1 .1,
0 4 -2 0,
0 0 3 2,
0 0 0 1
};
lbd12 = lbd12/2;
lambdat = lbd12'lbd12;

thetat = {
2 -1 -3 4,
-1 0 2 -3,
3 2 1 0
};
rankx = rows(lambdat);
rankz = rows(thetat);
@ Get pointer into stacked xy matrices @
b = cumsumc(yrows);
a = 1|(1+b[1:nobs-1]);
iptxy = a~b;

xdim = rankx - 1;
zdim = rankz - 1;
a = seqa(1,1,xdim);
xnames = 0 $+ "X " $+ ftocv(a,1,0);
```

```
a = seqa(1,1,zdim);
znames = 0 $+ "Z " $+ ftocv(a,1,0);
xynames = xnames|"Y ";

xdata = rndn(ntot,xdim);
xmat = ones(ntot,1)~xdata;
zdata = rndn(nobs,zdim);
zmat = ones(nobs,1)~zdata;
betat = zmat*thetat + rndn(nobs,rankx)*lbd12;
ydata = zeros(ntot,1);
@ Generate and store y data @
for i0 (1,nobs,1); i = i0;
xi = xmat[iptxy[i,1]:iptxy[i,2],.];
yi = xi*(betat[i,.]') + sigmat*rndn(yrows[i],1);
ydata[iptxy[i,1]:iptxy[i,2],.] = yi;
endfor;
xydata = xdata~ydata;
/*
*****************************************************
* Create & Read a Gauss file.  f1 is the file handle.
*****************************************************
*/
create f1 = xydata with ^xynames, 0, 8;
if writer(f1,xydata) /= rows(xydata);
errorlog "Conversion of XYDATA to Gauss File did not work";
endif;
closeall f1;
create f1 = zdata with ^znames, 0, 8;
if writer(f1,zdata) /= rows(zdata);
errorlog "Conversion of ZDATA to GAUSS File did not work";
endif;
closeall f1;
save yrows = yrows;
save sigmat = sigmat;
save betat = betat;
save thetat = thetat;
save lambdat = lambdat;
```

5.2 Bayesian Analysis

```
/*
**************************************************
* HBREG2.GSS
* HB Linear Regression Model.
* Allows for subject level design matrices.
*
* Y_i = X_i*beta_i + epsilon_i for i = 1,..., nsub
* epsilon_i is N(0,sigma^2 I)
*
* beta_i = Theta'z_i + delta_i
* delta_i is N(0,Lambda)
* B = Z*Theta + Delta
*
* PRIORS
* sigma^2 is Inverted Gamma(r0/2,s0/2)
* Theta is maxtrix normal(u0,v0).
* That is, vec(Theta) is N(vec(u0),v0).
* vec(theta) stacks the columns of theta.
* Lambda is Inverted Wishart(f0, g0)
**************************************************
*/

new;
outfile = "result.res"; @ Specify output file for saving results  @
@ outfile is a string variable that contains a file name  @
inxy  = "xydata"; @ Name of Gauss file with X,Y data @
inz   = "zdata"; @ Name of Gauss file with Z data @
flagtrue = 1; @ 1 -> knows true parameters from simulation @
/*
**************************************************
* Initialize parameters for MCMC
**************************************************
*/
smcmc = 100; @ number of iterations to save for analysis  @
skip = 1; @ Save every skip iterations @
nblow = 100; @ Initial transition iterations  @
nmcmc = nblow + skip*smcmc; @ total number of iterations @

/*
**************************************************
* Get data
**************************************************
*/
```

```
@ Input Gauss files @
open f1  = ^inxy; @ Get Gauss file for X, Y data  @
@ Opens Gauss file & assigns file handle f1 @
xydata = readr(f1,rowsf(f1));
@ readr reads in Gauss file with file handle f1. @
@ rowsf(f1) returns the number of rows in the Gauss file.  @
@ readr reads rowsf(f1) rows, which is the entir dataset. @
ci = close(f1);
xynames = setvars(inxy); @ Get the variable names that accompnay X, Y data @
ynames = xynames[rows(xynames)];
xnames = "CS"|xynames[1:rows(xynames)-1];

@ Last row of xydata is Y. @
xdata = xydata[.,1:cols(xydata)-1]; @ Get independent variables  @
ydata = xydata[.,cols(xydata)]; @ Get dependent variable @

open f1 = ^inz;
zdata = readr(f1,rowsf(f1));
ci = close(f1);
znames = setvars(inz);
znames = "CS"|znames;

loadm yrows = yrows; @ yrows gives the number of observations per subject @
nsub = rows(yrows); @ Number of subjects. @
ntot = sumc(yrows); @ Total number of observations. @
@ Create pointer based on yrows to access xdata and ydata @
b = cumsumc(yrows); @ cumsumc is the cumulative sum of yrows @
a = 1|(1+b[1:nsub-1]);
iptxy = a~b;
@ To use iptxy to get the design matrix and data for subject i: @
@ x_i = xdata[iptxy[i,1]:iptxy[i,2],.] @
@ y_i = ydata[iptxy[i,1]:iptxy[i,2]] @

xdim = cols(xdata);
zdim = cols(zdata);
@ add intercepts to xdata and zdata @
xdata  = ones(ntot,1)~xdata;
zdata = ones(nsub,1)~zdata;
rankx = cols(xdata);
rankz = cols(zdata);
thdim = rankx*rankz; @ dimension of vec(theta) @

@ Compute some sufficient statistics @
@ Get point to access stacked matrices of xtx, xtxi, and xty @
b = seqa(rankx,rankx,nsub);
```

```
a = 1|(1+b[1:nsub-1]);
iptxt = a~b;
bpool = invpd(xdata'xdata)*xdata'ydata;
xtx = zeros(rankx*nsub,rankx);
xty = zeros(rankx*nsub,1);
bhat = zeros(nsub,rankx); @ MLE of beta_i @
sse = 0;
for i0 (1,nsub,1); i = i0;
xi = xdata[iptxy[i,1]:iptxy[i,2],.];
yi = ydata[iptxy[i,1]:iptxy[i,2],.];
xitxi = xi'xi;
xityi = xi'yi;
xtx[iptxt[i,1]:iptxt[i,2],.] = xitxi;
xty[iptxt[i,1]:iptxt[i,2],.] = xityi;
if rank(xitxi) >= rankx; @ Got an inverse @
xitxii = invpd(xitxi);
bhat[i,.] = (xitxii*xityi)';
else; @ Use the pooled estimate @
bhat[i,.] = bpool';
endif;
resid = yi - xi*(bhat[i,.]');
sse = sse + resid'resid;
endfor;
s2hat = sse/ntot; @ MLE of sigma2 @
sighat = sqrt(s2hat);
ztz = zdata'zdata;



/*
************************************************
* Initialize Priors
************************************************
*/

@ Prior for sigma2 is IG(r0/2, s0/2) @
r0  = 2; s0 = 2;
rn  = r0 + ntot;


@ Prior for theta is N(u0,v0) @

u0  = zeros(thdim,1);
v0    = 100*eye(thdim); @ thdim = rankx*rankz    @
v0i   = invpd(v0);     @ used in updating theta @
```

```
v0iu0  = v0i*vec(u0);          @ used in updating theta @

@ Lambda^{-1} is W_rankx(f0,g0 ) @
@ f0 = prior df, g0 = prior scale matrix @
f0  = rankx+2;  f0n = f0 + nsub;
g0i  = eye(rankx);   @ g0^{-1} @

/*
************************************************
* Initialize MCMC
************************************************
*/

beta  = zeros(nsub, rankx);
sigma  = 1;
sigma2 = s2hat;
theta = zeros(rankz,rankx);
lambda = eye(rankx);
lambdai = invpd(lambda);

@ Define data structures for saving iterates & computing posterior means & std @
betam = zeros(nsub,rankx); @ posterior mean of beta  @
betas = zeros(nsub,rankx); @ posterior std of beta  @
sigmag = zeros(smcmc,1);
thetag = zeros(smcmc,thdim);
c = rankx*(rankx+1)/2;
lambdag = zeros(smcmc,c); @ save unique elements of lambda @


/*
************************************************
* Do MCMC
************************************************
*/
@ Do the initial transition period @
for i1 (1,nblow,1); imcmc = i1;
call gethbreg;
endfor;

for i1 (1,smcmc,1); imcmc = i1; @ Save smcmc iterations  @
for i2 (1,skip,1); jmcmc = i2; @ Save every skip iterations  @
call gethbreg;
endfor;
sigmag[imcmc] = sigma;
thetag[imcmc,.] = vecr(theta)';
```

```
betam = betam + beta;
betas = betas + beta^2;
lambdag[imcmc,.] = vech(lambda)'; @ vech gets unique elements of symmetric matrix @
endfor;


/*
**********************************************
* Compute Posterior Means and STD
**********************************************
*/
@ Compute Posterior Means @
betam = betam/smcmc;
sigmam = meanc(sigmag);
thetam = meanc(thetag);
thetam = reshape(thetam,rankz,rankx);
lambdam = xpnd(meanc(lambdag)); @ xpnd is opposite of vech @

@ Compute Posterior STD @
betas = sqrt( abs(betas - smcmc*betam^2)/smcmc);
sigmas = stdc(sigmag);
thetas = stdc(thetag);
thetas = reshape(thetas, rankz, rankx);
lambdas = xpnd(stdc(lambdag));

@ Predict yi @
yhhb = zeros(ntot,1);
yhml = yhhb;
for i0 (1, nsub, 1); i = i0;
xi = xdata[iptxy[i,1]:iptxy[i,2],.];
yhhb[iptxy[i,1]:iptxy[i,2]] = xi*(betam[i,.]');
yhml[iptxy[i,1]:iptxy[i,2]] = xi*(bhat[i,.]');
endfor;
cr = corrx(ydata~yhhb);
hbr = cr[1,2];
estd = ydata - yhhb;
estd = sqrt(estd'estd/ntot);
cr = corrx(ydata~yhml);
mlr = cr[1,2];
estdml  =  ydata - yhml;
estdml  = sqrt(estdml'estdml/ntot);


/*
**********************************************
```

```
* Do some output
*********************************************
*/
call outputanal;


@ Plot saved iterations against iterations number @
t  = seqa(nblow+skip,skip,smcmc); @ saved iteration number @
title("Error STD versus Iteration");
xy(t,sigmag);
title("Theta versus Iteration");
xy(t,thetag);
title("Lambda versus Iteration");
xy(t,lambdag);
title("MLE & HB for " $+ xnames[2]$+ " versus " $+ xnames[1]);
_plctrl = -1;
xy(bhat[.,1]~betam[.,1],bhat[.,2]~betam[.,2]);
graphset; @ Return to default settings @


end;

/*
*********************************************
* GETHBREG
* Does one iteration of the HB regression model.
* INPUT
* Global Variables
* OUTPUT
* Global Variables
*********************************************
*/
PROC (0) = gethbreg;
local vibn, vibn12, ebin, yhat, sse, sn, resid, gni, gn, gn12, w, i0, i, limub, yhati,
betai;

limub = zdata*theta*lambdai; @ Used in posterior mean of beta @
/*
*****************************************
* Generate beta
* beta_i is N(mbin, vbn)
* vbn  = ( X_i'X_i/sigma2 + Lambda^{-1} }^{-1}
* mbin = vbn*( X_i'Y_i/sigma2 + Lambda^{-1}*Theta*Z_i)
*****************************************
*/
```

```
sse = 0;
for i0 (1,nsub,1); i = i0;
xi  = xdata[iptxy[i,1]:iptxy[i,2],.];
yi  = ydata[iptxy[i,1]:iptxy[i,2],.];
xitxi  = xtx[iptxt[i,1]:iptxt[i,2],.];
xityi  = xty[iptxt[i,1]:iptxt[i,2],.];
vibn = xitxi/sigma2 + lambdai;
vibn12 = chol(vibn);
ebin = xityi/sigma2 + limub[i,.]';
betai = cholsol(ebin + vibn12'rndn(rankx,1), vibn12);
beta[i,.] = betai';
yhati = xi*betai;
resid = yi - yhati;
sse = sse + resid'resid;
endfor;

/*
*******************************************
* Generate sigma2
* sigma2 is IG(rn/2, sn/2)
* rn = r0 + ntot
* sn = s0 + sum_{i=1}^{nsub} (Y_i - X*beta_i)'(Y_i - X*beta_i)
*******************************************
*/
sn = s0 + sse;
sigma2 = sn/(2*rndgam(1,1,rn/2));
sigma = sqrt(sigma2);

/*
*****************************************
* Generate Theta and Lambda from multivariate model:
* B = Z*Theta + N(0,Lambda)
*****************************************
*/
{theta, lambda, lambdai} =
getmulreg(beta,zdata,ztz,theta,lambda,lambdai,v0i,v0iu0,f0n,g0i);


endp;



/*
*******************************************
* GETMULREG
```

```
* Generate multivariate regression parameters.
* Yd = Xd*parmat + epsilon
*
* INPUT
* yd = dependent variables
* xd = independet variables
* xdtxd = xd'xd
*
* parmat = current value of coefficient matrix
* var = current value of covariance matrix
* vari = its inverse
* v0i = prior precisions for bmat
* v0iu0 = prior precision*prior mean for bmat
* f0n = posterior df for sigma
* g0i = prior scaling matrix inverse for sigma

*
* OUTPUT
* parmat = updated rankx x mvar coefficient matrix
* var = updated variance
* vari = updated inverse of sigma
*
* Calling Statement:
{parmat, var, vari} = getmulreg(yd,xd,xdtxd,parmat,var,vari,v0i,v0iu0,f0n,g0i);
*******************************************
*/
PROC (3) = getmulreg(yd,xd,xdtxd,parmat,var,vari,v0i,v0iu0,f0n,g0i);
local vb12, ubn, par, pdim, resid, gni, gn, rp, cp;

rp = rows(parmat);
cp = cols(parmat);
pdim = rp*cp;

/*
*******************************************
* Generate parmat from N_{rp x cp}(M,v)
* par = vecr(parmat)
* par is N(u,V) whee u = vec(M');
* V = (Xd'Xd.*.Var^{-1} + V_0^{-1})^{-1}
* u = V*( (Xd'.*.Var^{-1})*vec(Yd') + V_0^{-1}u_0 )
*******************************************
*/

vb12 = chol(xdtxd.*.vari + v0i);
ubn = ( (xd').*.vari )*vecr(yd) + v0iu0;
```

```
par = cholsol(ubn + vb12'rndn(pdim,1), vb12);
parmat = reshape(par,rp,cp);

/*
****************************************
* Generate Var
*  Var^{-1} is Wishart df = f0n, scale matrix = gn
****************************************
*/
resid  = yd - xd*parmat;
gni = g0i + resid'resid;
gn   = invpd(gni);
{vari, var}  = wishart(cp,f0n,gn);

retp(parmat,var,vari);
endp;



/*
**************************************************************************
*  OUTPUTANAL
* Does analysis of output and save some results
**************************************************************************
*/
PROC (0) = outputanal;
local  bout, sout, ebeta, sbeta, cb, rmse, fmtn1, fmtn2, fmts1, fmts2,  a, b,
betat, sigmat, thetat, lambdat, bratio ;

if flagtrue == 1; @ Did a simulation @
load betat = betat;
load sigmat = sigmat;
load thetat = thetat;
load lambdat = lambdat;
endif;

@ Define formats for fancy printing @
@ Used to print a matrix of alpha & numeric variables @

format 10,5; @ Default pring format @
let fmtn1[1,3] = "*.*lf" 10 5; @ Format for printing numeric variable  @
let fmtn2[1,3] = "*.*lf" 10  0; @ Format for numeric variable, no decimal @

let fmts1[1,3] = "-*.*s" 10 9; @ Format for alpha, left justify   @
let fmts2[1,3]  = "*.*s" 10 9; @ Format for alpha, right justify     @
```

```
output file = ^outfile reset; @ outfile is the file handle for the output file @
@ Route printed output to the defined by outfile  @
print "Results from HBReg2.GSS";
print "Hierarchical Bayesian linear regression model using MCMC.";
print "Different design matrices for each subject";
print "Y_i  = X_i*beta_i + epsilon_i";
print "beta_i  = Theta'z_i + delta_i";
print "or";
print "B = Z*Theta + Delta";
print;
print "epsilon_i  is N(0,sigma2*I)";
print "delta_i  is N(0, Lambda)";
print;
print "Ouput file: " getpath(outfile); @ File assigned to file handle outfile @
datestr(date); @ Print the current data @
print;
print;
print "-------------------------------------------";
print;
print "MCMC Analysis";
print;
print "Total number of MCMC iterations              = " nmcmc;
print "Number of iterations used in the analysis    = " smcmc;
print "Number in transition period                  = " nblow;
print "Number of iterations between saved iterations = " skip-1;
print;
print "Number of subjects                 = " nsub;
print "Mean # of observations per subject  = " meanc(yrows);
print "STD  # of observations per subject  = " stdc(yrows);
print "MIN  # of observations per subject  = " minc(yrows);
print "MAX  # of observations per subject  = " maxc(yrows);
print "Total number of observations       = " ntot;
print "Number of dependent variables X    = " xdim " (excluding intercept)";
print "Number of dependent variables Z    = " zdim " (excluding intercept)";
print;
print "Dependent variable is " $ynames;
print;
print "Independent variables in first level equation: Y_i = X_i*beta_i + epsilon_i";
print "       Summary Statistics for X";
call sumstats(xnames,xdata,fmts1,fmts2,fmtn1);
print;
print "Independent variables in second level equation: beta_i = Theta*z_i + delta_i";
print "      Summary Statistics for Z";
print sumstats(znames,zdata,fmts1,fmts2,fmtn1);
print;
```

```
print "-------------------------------------------";
print;
print "Fit Statistics for ML";
print "Muptilpe R            = " mlr;
print "R-Squared             = " mlr^2;
print "Error STD DEV         = " estdml;
print;
print "Fit Statistics for HB";
print "Muptilpe R            = " hbr;
print "R-Squared             = " hbr^2;
print "Error STD DEV         = " estd;
print;
print "-------------------------------------------";
print "Estimation of the error STD sigma";
if flagtrue == 1;
print "True Sigma     = " sigmat;
endif;
print "MLE            = " sighat;
print "Posterior Mean = " sigmam;
print "Posterior STD  = " sigmas;
print;
print "-------------------------------------------";
print;
print "ML Pooled Estimate of Beta";
bout = xnames~bpool;
call outmat(bout,fmts1,fmtn1);
print;
print "-------------------------------------------";
print;
print "Statistics for Individual-Level Regression Coefficients";
if flagtrue == 1;
ebeta = meanc(betat);
sbeta = stdc(betat);
print "True Beta";
sout = {"Variable" "Mean" "STD"};
call outitle(sout,fmts1,fmts2);
bout = xnames~ebeta~sbeta;
call outmat(bout,fmts1,fmtn1);
endif;

print "MLE of Beta";
sout = {"Variable" "MeanMLE"  "StdMLE" };
call outitle(sout,fmts1,fmts2);
bout = xnames~meanc(bhat)~stdc(bhat);
call outmat(bout,fmts1,fmtn1);
```

```
print "HB Estimates of Beta";
ebeta = meanc(betam);
sbeta = sqrt( meanc( (betas^2)) + stdc( betam)^2);
sout = {"Variable" "PostMean" "PostSTD" };
call outitle(sout,fmts1,fmts2);
bout = xnames~ebeta~sbeta;
call outmat(bout,fmts1,fmtn1);
print;
print;
print "Proportion of times that |Post Mean Beta|/(Post STD) > 2.";
bratio  = betam./betas;
b = bratio .> 2;
bout = xnames~(meanc(b));
call outmat(bout,fmts1,fmtn1);
print;
print "-------------------------------------------";
print;
if flagtrue == 1;
print "Comparison of True Beta to Individual Level Estimates";
for i0 (1,rankx,1); i = i0;
print "Component " i;
cb  = corrx( betat[.,i]~betam[.,i] );
rmse  = betat[.,i] - betam[.,i];
rmse = rmse'rmse;
rmse = sqrt(rmse/nsub);
print "Correlation between true and HB  = " cb[1,2];
print "RMSE between true and HB         = " rmse;
print;
cb  = corrx( betat[.,i]~bhat[.,i] );
rmse  = betat[.,i] - bhat[.,i];
rmse = rmse'rmse;
rmse = sqrt(rmse/nsub);
print "Correlation between true and MLE = " cb[1,2];
print "RMSE between true and MLE        = " rmse;
print;
endfor;
endif;
print "-------------------------------------------";
print;
print "HB Estimates of Theta";
sout = "  "~(xnames');
if flagtrue == 1;
print "True Theta";
call outitle(sout,fmts1,fmts2);
```

```
bout = znames~thetat;
call outmat(bout,fmts1,fmtn1);
print;
endif;
print "Posterior Mean of Theta";
print outitle(sout,fmts1,fmts2);
bout = znames~thetam;
call outmat(bout,fmts1,fmtn1);
print;
print "Posterior STD of Theta";
call outitle(sout,fmts1,fmts2);
bout = znames~thetas;
call outmat(bout,fmts1,fmtn1);
print;
print "Post Mean/Post STD";
print outitle(sout,fmts1,fmts2);
bout = znames~(thetam./thetas);
call outmat(bout,fmts1,fmtn1);
print;
print "-----------------------------------------";
print;
sout = "  "~(xnames');
print "HB Estimate of Lambda";
if flagtrue == 1;
print "True Lambda";
call outitle(sout,fmts1,fmts2);
bout = xnames~lambdat;
call outmat(bout,fmts1,fmtn1);
print;
endif;
print "Posterior Mean of Lambda";
call outitle(sout,fmts1,fmts2);
bout = xnames~lambdam;
call outmat(bout,fmts1,fmtn1);
print;
print "Posterior STD of Lambda";
call outitle(sout,fmts1,fmts2);
bout = xnames~lambdas;
call outmat(bout,fmts1,fmtn1);
print;
print "=======================================";

output off;
closeall;
endp;
```

```
/*
****************************************************************
* OUTITLE
* Prints header for columns of numbers.
* INPUT
* a  = character row vector of column names
* fmts1 = format for first column
* fmts2 = format for second column
* OUTPUT
* None
****************************************************************
*/
PROC (0) = outitle(a,fmt1,fmt2);
local mask, fmt, flag, ncols;
ncols = cols(a);
mask = zeros(1,ncols);
fmt = fmt1|(ones(ncols-1,1).*.fmt2);
flag = printfm(a,mask,fmt);
print;
endp;
/*
************************************************************
* OUTMAT
* Outputs a matrix:
* (Character Vector)~(Numeric matrix);
* The entries in the numeric matrix have the same format
* INPUT
* bout = matrix to be printed
* fmts = format for string
* fmtn = format for numeric matrix
* OUTPUT
* None
****************************************************************
*/
PROC (0) = outmat(bout,fmts,fmtn);
local fmt,mask,flag,ncols, nrows;
ncols = cols(bout);
nrows = rows(bout);
fmt = fmts|(ones(ncols-1,1).*.fmtn);
mask = zeros(nrows,1)~ones(nrows,ncols-1);
flag = printfm(bout,mask,fmt);
```

```
print;
endp;

/*
****************************************************************
* SUMSTATS
* Prints summary statistics for a data matrix
* INPUT
* names = charater vector of names
* data = data matrix
* fmts1 = format for string
* fmts2 = format for string
* fmtn = format for numbers
* OUTPUT
* None
****************************************************************
*/
PROC (0) = sumstats(names,data,fmts1,fmts2,fmtn);
local a, bout;
a = {"Variable" "Mean" "STD" "MIN" "MAX"};
call outitle(a,fmts1,fmts2);
bout = names~meanc(data)~stdc(data)~minc(data)~maxc(data);
call outmat(bout,fmts1,fmtn);
endp;
```

Chapter 6

HB Regression: Mixture Model

6.1 Data Generation

```
/*
***********************************************
*  GMIXREG.GSS
* Generats data for HB Regression Model
* Y_i  = X_i*beta_i + epsilon_i
* beta_i = sum_{k=1}^K N(beta_i|theta_k,Lambda_k)
* epsilon_i is N(0,sigma2*I)
* Different Design Matrices
***********************************************
*/
new;
nsub = 200; @ Number of subjects  @
yrows = 3 + ceil(7*rndu(nsub,1)); @ Number of observations per subject @
ntot = sumc(yrows);
sigmat = 5; @ True error STD @
rankx = 2; @ rank(X_i) @
mmodt = 3; @ Three mixture componts @
@ thetat[.,j] is the mean for component j @
thetat  = {     0    -10     7,
                0      7     5  };
@ lambdat stacks 3 covariance matrics, each is a 2 by 2 matrix. @
lambdat = {  1    0,
             0    1,
             25   9,
             9    4,
             9   -5,
            -5    5   };



@ Get pointer into stacked xy matrices @
b = cumsumc(yrows);
a = 1|(1+b[1:nsub-1]);
iptxy = a~b;

@ Get pointer into sacked lambdat @
b = seqa(rankx,rankx,mmodt);
a = 1|(1+b[1:mmodt-1]);
iptgp = a~b;


xdim = rankx - 1;
a = seqa(1,1,xdim);
```

```
xnames = 0 $+ "X " $+ ftocv(a,1,0);
xynames = xnames|"Y ";

xdata = rndn(ntot,xdim);
xmat = ones(ntot,1)~xdata;
psit    = { 0.2,  0.3,  0.5 }; @ Mixture probabilities  @
zprob   = ones(nsub,1).*.(psit');
z       = rndzmn(zprob); @ Z gives class membership  @

@ Compute Chol of lambdat @
lmbd12 = lambdat;
for i0 (1,mmodt,1); i = i0;
lmbd12[iptgp[i,1]:iptgp[i,2],.] = chol(lambdat[iptgp[i,1]:iptgp[i,2],.]);
endfor;

@ Generate Beta and Ydata @
betat = zeros(rankx,nsub);
ydata = zeros(ntot,1);

for i0 (1,nsub,1); i = i0;
betai = thetat[.,z[i]] + lmbd12[iptgp[z[i],1]:iptgp[z[i],2],.]'rndn(rankx,1);
betat[.,i] = betai;
xi = xmat[iptxy[i,1]:iptxy[i,2],.];
yi = xi*betai + sigmat*rndn(yrows[i],1);
ydata[iptxy[i,1]:iptxy[i,2],.] = yi;
endfor;
xydata = xdata~ydata;
/*
*****************************************************
* Create a Gauss file.  f1 is the file handle.
* The Gauss file will be called "XYDATA."
* The column will be named by the strings in the character array xyname.
* ^xyname means use the names in the character string.
* 0, 8 gives double precision real numbers.
*****************************************************
*/
create f1 = xydata with ^xynames, 0, 8;
/*
*****************************************************
* Next read data into the Gauss file by using the writer command.
* f1 is the file handle defined in previous command.
* xydata is the data matrix that we just created.
* writer returns the number of rows read to f1.
* If it is not rows(xydata), something bad happended.
*****************************************************
```

```
*/
if writer(f1,xydata) /= rows(xydata);
errorlog "Conversion of XYDATA to Gauss File did not work";
endif;
closeall f1;
save yrows = yrows;
save classt = z; @ true classifications @
save sigmat = sigmat;
save betat = betat;
save thetat = thetat;
save lambdat = lambdat;
save psit = psit;


@ Compute individual level MLEs @

bhat = zeros(rankx,nsub);
sse = 0;
for i0 (1,nsub,1); i = i0;
xi = xmat[iptxy[i,1]:iptxy[i,2],.];
yi = ydata[iptxy[i,1]:iptxy[i,2],.];
xitxi = xi'xi;
xitxii = invpd(xitxi);
xityi = xi'yi;
bhat[.,i] = xitxii*xityi;
resid = yi - xi*bhat[.,i];
sse = sse + resid'resid;
endfor;
s2hat = sse/ntot; @ MLE of sigma2 @
sighat = sqrt(s2hat);


@ Do some plots @
_plctrl = -1;
title("Y versus X");
xy(xydata[.,1],xydata[.,cols(xydata)]);
title("True Slope verus Intercept");
xy(betat[1,.]',betat[2,.]');
title("True & MLE Slope versus Intercept");
xy(betat[1,.]'~bhat[1,.]',betat[2,.]'~bhat[2,.]');
graphset;
```

6.2 Bayesian Analysis

```
/*
**************************************************
* MixReg.GSS
* HB Linear Regression Model.
* Heterogeneity for subject level parameters have a mixture of normals
* Allows for subject level design matrices.
*
* Y_i = X_i*beta_i + epsilon_i for i = 1,..., nsub
* epsilon_i is N(0,sigma^2 I)
*
* beta_i = sum_{m=1}^M psi_m N(beta_i|theta_m,Lambda_n)
* 0 < psi_1 < psi_2 .. < psi_M and sum_{m=1}^M psi_m = 1.
* delta_i is N(0,Lambda)
*
* PRIORS
* sigma^2 is Inverted Gamma(r0/2,s0/2)
* theta_m is normal(u0,v0).
* Lambda_m is Inverted Wishart(f0, g0)
*
*---->Note:
* This program saves subject-level beta_i as columns, not rows.
* beta is a nsub by rankx matrix
*
**************************************************
*/
new;
mmod = 3; @ mmod = number of mixture components @
outfile = "results1.dat"; @ Specify output file for saving results  @
@ outfile is a string variable that contains a file name  @
inxy  = "xydata"; @ Name of Gauss file with X,Y data @
flagtrue = 1; @ 1 -> knows true parameters from simulation @

pcount  = 5;    @ max number of times that assignments of subjects  @
@ to groups can conflict with ordering: n_1 <= ... <= n_k @
@ where n_j = number of subjects in each group.          @
pflag = 0; @ flag for runs in viloations @
@ pflag = 0 -> last iterations not violation @
@ pflag = 1 -> last iteration is a violation @


/*
**************************************************
* Initialize parameters for MCMC
```

```
************************************************
*/
smcmc = 100; @ number of iterations to save for analysis  @
skip = 1; @ Save every skip iterations @
nblow = 100; @ Initial transition iterations  @
nmcmc = nblow + skip*smcmc; @ total number of iterations @


mint  = seqa(1,1,mmod);
/*
************************************************
* Get data
************************************************
*/
@ Input Gauss files @
open f1  = ^inxy; @ Get Gauss file for X, Y data  @
@ Opens Gauss file & assigns file handle f1 @
xydata = readr(f1,rowsf(f1));
@ readr reads in Gauss file with file handle f1. @
@ rowsf(f1) returns the number of rows in the Gauss file.  @
@ readr reads rowsf(f1) rows, which is the entir dataset. @
ci = close(f1);
xynames = setvars(inxy); @ Get the variable names that accompnay X, Y data @
ynames = xynames[rows(xynames)];
xnames = "Constant"|xynames[1:rows(xynames)-1];

@ Last row of xydata is Y. @
xdata = xydata[.,1:cols(xydata)-1]; @ Get independent variables  @
ydata = xydata[.,cols(xydata)]; @ Get dependent variable @


loadm yrows = yrows; @ yrows gives the number of observations per subject @
nsub = rows(yrows); @ Number of subjects. @
ntot = sumc(yrows); @ Total number of observations. @
@ Create pointer based on yrows to access xdata and ydata @
b = cumsumc(yrows); @ cumsumc is the cumulative sum of yrows @
a = 1|(1+b[1:nsub-1]);
iptxy = a~b;
@ To use iptxy to get the design matrix and data for subject i: @
@ x_i = xdata[iptxy[i,1]:iptxy[i,2],.] @
@ y_i = ydata[iptxy[i,1]:iptxy[i,2]] @

xdim = cols(xdata);
@ add intercepts to xdata  @
xdata  = ones(ntot,1)~xdata;
```

```
rankx = xdim+1;
@ Create pointer to get into stacked lambda matrix @
if mmod > 1;
b = seqa(rankx,rankx,mmod);   @ {rankx, 2*rankx, ..., mmod*rankx } @
a = 1|(1+b[1:mmod-1]); @ {1, rankx+1, 2*rankx+1, ..., (mmod-1)*rankx+1} @
iptgp = a~b;
else;
iptgp = 1~rankx;
endif;


@ Compute some sufficient statistics @
@ Get point to access stacked matrices of xtx, xtxi, and xty @
b = seqa(rankx,rankx,nsub);
a = 1|(1+b[1:nsub-1]);
iptxt = a~b;
xtx = zeros(rankx*nsub,rankx);
xty = zeros(rankx*nsub,1);
bhat = zeros(rankx,nsub); @ MLE of beta_i @
sse = 0;
for i0 (1,nsub,1); i = i0;
xi = xdata[iptxy[i,1]:iptxy[i,2],.];
yi = ydata[iptxy[i,1]:iptxy[i,2],.];
xitxi = xi'xi;
xitxii = invpd(xitxi);
xityi = xi'yi;
xtx[iptxt[i,1]:iptxt[i,2],.] = xitxi;
xty[iptxt[i,1]:iptxt[i,2],.] = xityi;
bhat[.,i] = xitxii*xityi;
resid = yi - xi*bhat[.,i];
sse = sse + resid'resid;
endfor;
s2hat = sse/ntot; @ MLE of sigma2 @
sighat = sqrt(s2hat);



/*
************************************************
* Initialize Priors
************************************************
*/

@ Prior for sigma2 is IG(r0/2, s0/2) @
```

```
r0  = 2; s0 = 2;
rn  = r0 + ntot;

@ {psi_m}, the mixture probabilities, have ordered Dirchlet proir. @
w0    = ones(mmod,1);
xgam = seqa(1,1,mmod)'; @ used in generating ordered dirichlet @

@ Prior for theta_i is N(u0,v0) @

u0  = zeros(rankx,1);
v0    = 100*eye(rankx);
v0i   = invpd(v0);     @ used in updating theta @
v0iu0  = v0i*vec(u0);         @ used in updating theta @

@ Lambda^{-1} is W_rankx(f0,g0 ) @
@ f0 = prior df, g0 = prior scale matrix @
f0  = rankx+2;
g0i  = eye(rankx);   @ g0^{-1} @

/*
************************************************
* Initialize MCMC
************************************************
*/
psi = mint/sumc(mint); @ Class probabilities @
@ Assign membership based on random start                  @
zprob = ones(nsub,1).*.psi'; @ Individual level prob.  .*. is Kronecker product. @
z  = rndzmn(zprob); @ Generate multinomials @
classn  = sumc(z .== mint' ); @ number in each of the mmod classes @
rclass  = rankindx(classn,1);

beta = bhat;
theta = zeros(rankx,mmod);
lambda = zeros(rankx*mmod,rankx);
lambdai = lambda;
bvars = zeros(rankx*mmod,1);
for i0 (1,mmod,1); i = i0;
if classn[i] > rankx+1; @ We have some observations in this class @
zi = z .== i;
bi = selif(beta',zi)'; @ Get beta for subjects assigned to group i @
mbi = meanc(bi');
resid = bi - mbi;
lbdai = resid*resid'/classn[i];
else;
mbi = zeros(rankx,1);
```

```
lbdai = eye(rankx);
endif;
theta[.,i]  = mbi;
lambda[iptgp[i,1]:iptgp[i,2],.]  = lbdai;
lbda = invpd(lbdai);
lambdai[iptgp[i,1]:iptgp[i,2],.]  = lbda;
bvars[iptgp[i,1]:iptgp[i,2]] = diag(lbda);
endfor;
sigma2 = s2hat;
sigma = sqrt(sigma2);

/*
*********************************************************
* Arrays for saving iterations & computing posterior means & std
*********************************************************
*/

betam = zeros(rankx,nsub); @ posterior mean of beta  @
betas = zeros(rankx,nsub); @ posterior std of beta  @
sigmag = zeros(smcmc,1); @ MCMC iterations for error std @
psig = zeros(smcmc,mmod);
zprobm = zeros(nsub,mmod); @ Individual level class membership probabilities @
thdim = rankx*mmod;
thetag = zeros(smcmc,thdim);
lambdag = zeros(smcmc,mmod*rankx); @ MCMC iterations for heterogeniety variance @
lambdam = zeros(rankx*mmod,rankx);
lambdas = lambdam;
loglikeg = zeros(smcmc,1);
loglike = 0;
/*
************************************************
* Do MCMC
************************************************
*/
@ Do the initial transition period @
nmcmc = 0; @ Just a counter @
ipcount = 0; @ Counter of order restriction violations @
for i1 (1,nblow,1); imcmc = i1;
nmcmc = nmcmc + 1;
call getmixreg;
endfor;

for i1 (1,smcmc,1); imcmc = i1; @ Save smcmc iterations  @
for i2 (1,skip,1); jmcmc = i2; @ Save every skip iterations  @
nmcmc = nmcmc + 1;
```

```
call getmixreg;
endfor;
sigmag[imcmc] = sigma;
thetag[imcmc,.] = vec(theta)';
psig[imcmc,.] = psi';
betam = betam + beta;
betas = betas + beta^2;
zprobm = zprobm + zprob;
lambdam = lambdam + lambda;
lambdas = lambdas + lambda^2;
lambdag[imcmc,.] = bvars';
loglikeg[imcmc,.] = loglike;
endfor;


/*
***********************************************
* Compute Posterior Means and STD
***********************************************
*/
@ Compute Posterior Means @
betam = betam/smcmc;
zprobm = zprobm/smcmc;
sigmam = meanc(sigmag);
psim = meanc(psig);
thetam = meanc(thetag);
thetam = reshape(thetam,mmod,rankx)';
lambdam = lambdam/smcmc;
loglikem = meanc(loglikeg);

@ Compute Posterior STD @
betas = sqrt( abs(betas - smcmc*betam^2)/smcmc);
sigmas = stdc(sigmag);
psis = stdc(psig);
thetas = stdc(thetag);
thetas = reshape(thetas, mmod, rankx)';
lambdas = sqrt( abs(lambdas - smcmc*lambdam^2)/smcmc);
loglikes = stdc(loglikeg);

@ Predict yi @
yhat = zeros(ntot,1);
br = zeros(nsub,1); @ multiple R for each subject @
estd = br; @ error std @
for i0 (1, nsub, 1); i = i0;
xi = xdata[iptxy[i,1]:iptxy[i,2],.];
```

```
yi = ydata[iptxy[i,1]:iptxy[i,2]];
yhati = xi*betam[.,i];
yhat[iptxy[i,1]:iptxy[i,2]] = yhati;
cy = corrx(yi~yhati); @ correlation matrix of yi and yhati @
br[i] = cy[1,2];
resid = yi - yhati;
estd[i] = sqrt(resid'resid/yrows[i]);
endfor;


/*
*******************************************
* Do some output
*******************************************
*/
call outputanal;


@ Plot saved iterations against iterations number @
t  = seqa(nblow+skip,skip,smcmc); @ saved iteration number @
title("Error STD versus Iteration");
xy(t,sigmag);
title("Mixture Probabilities versus Iteration");
xy(t,psig);
title("Heterogeneity Means versus Iteration");
xy(t,thetag);
title("Heterogeneity VARS versus Iteration");
xy(t,lambdag);
_plctrl = -1;
if flagtrue == 1;
load betat  = betat;
load sigmat = sigmat;
load psit = psit;
load thetat = thetat;
load lambdat = lambdat;
title("True & HB Slope vs Intercept");
xy(betat[1,.]'~betam[1,.]',betat[2,.]'~betam[2,.]');
endif;
title("HB & ML Slope vs Intercept");
xy(betam[1,.]'~bhat[1,.]',betam[2,.]'~bhat[2,.]');
graphset;


end;
```

```
/*
*********************************************
* GETMIXREG
* Does one iteration of the HB regression model.
* INPUT
* Global Variables
* OUTPUT
* Global Variables
*********************************************
*/
PROC (0) = getmixreg;
local
sse, i0,i,thetak,lambdaik,xi,yi,xitxi,xityi,vibn,vibn12,ebin,yhati,resid,sn,
fk,k,betak,mbetak,c,b,fnk,gnki,gnk,gnk12,w,lambdak,resid2,dlambik, zmaxp,
rclass, pflag,  lamb0,lamb2,j;


zprob = zeros(nsub,mmod); @ used in computing the P(z[i] = k) @

/*
*****************************************
* Generate beta_i
* If Y_i belongs to class k, then
* beta_i is N(mbin, vbn)
* vbn  = ( X_i'X_i/sigma2 + Lambda_k^{-1} }^{-1}
* mbin = vbn*( X_i'Y_i/sigma2 + Lambda_k^{-1}*Theta_k)
*****************************************
*/
sse = 0;
for i0 (1,nsub,1); i = i0;
thetak = theta[.,z[i]];
lambdaik = lambdai[iptgp[z[i],1]:iptgp[z[i],2],.];
xi  = xdata[iptxy[i,1]:iptxy[i,2],.];
yi  = ydata[iptxy[i,1]:iptxy[i,2],.];
xitxi  = xtx[iptxt[i,1]:iptxt[i,2],.];
xityi  = xty[iptxt[i,1]:iptxt[i,2],.];
vibn = xitxi/sigma2 + lambdaik;
vibn12 = chol(vibn);
ebin = xityi/sigma2 + lambdaik*thetak;
beta[.,i] = cholsol(ebin + vibn12'rndn(rankx,1), vibn12);
yhati = xi*beta[.,i];
resid = yi - yhati;
sse = sse + resid'resid;
endfor;
```

```
/*
*********************************************
* Generate sigma2
* sigma2 is IG(rn/2, sn/2)
* rn = r0 + ntot
* sn = s0 + sum_{i=1}^{nsub} (Y_i - X*beta_i)'(Y_i - X*beta_i)
*********************************************
*/
sn = s0 + sse;
sigma2 = sn/(2*rndgam(1,1,rn/2));
sigma = sqrt(sigma2);

@ Compute log likelihood @
loglike = -(ntot*ln(sigma2) + sse/sigma2)/2;
/*
*********************************************
*  Generate theta_k and lambda_k
*********************************************
*/
for fk (1,mmod,1); k = fk;
@ do we have observations in class k? @
if classn[k] > 0.5;
betak      = (selif(beta', z .== k))';          @ beta with z=k@
mbetak     = meanc(betak');
else;
mbetak     = zeros(rankx,1);
endif;
lambdaik   = lambdai[iptgp[k,1]:iptgp[k,2],.];

/*
*********************************************
*  Generate theta_k given z, beta etc.
*  theta_k is N(u_nk,v_nk)
*  v_n,k = (n_k lambda_k^{-1} + v_0,k^{-1})
*  u_n,k = v_n,k(n_k lambda_k^{-1} meanc(beta_k) + v_0,k^{-1} u_0,k
*********************************************
*/
c  = chol(classn[k]*lambdaik + v0i);
b  = classn[k]*lambdaik*mbetak + v0iu0;
thetak = cholsol(b+c'rndn(rankx,1),c);
theta[.,k] = thetak;
/*
*********************************************
*  Generate Lambda_k from IW_p(fnk, Gnk)
*    fnk = f0k + classn_k
```

```
*    Gnk =
* (G0k^{-1} +  sum I(z_i=k)(beta_i-theta_k)(beta_i-theta_k)')^{-1}
*******************************************
*/
if classn[k] > 0.5;    @ observations in class k @
fnk   = f0 + classn[k];
resid = betak - theta[.,k];
gnki  = g0i + resid*resid';
else; @ no observations, no updating of prior @
fnk   = f0;
gnki  = g0i;
endif;
gnk   = invpd(gnki);
{lambdaik, lambdak}  = wishart(rankx,fnk,gnk);

dlambik  = det(lambdaik);          @ determinant of lambdaik @
@ store lambda's  @
lambdai[iptgp[k,1]:iptgp[k,2],.]  = lambdaik;
lambda[iptgp[k,1]:iptgp[k,2],.]   = lambdak;
bvars[iptgp[k,1]:iptgp[k,2]] = diag(lambdak); @ Save diagonals for plotting @

@ Compute the probability of class k for all subjects @
if mmod > 1;
@ Start computing P(z[i] = k) @
zprob[.,k] = 0.5*ln(dlambik)*ones(nsub,1);
resid = beta - thetak;
resid2 = lambdaik*resid;
for i0 (1,nsub,1); i = i0;
zprob[i,k] = zprob[i,k] - 0.5*resid[.,i]'resid2[.,i];
endfor;
endif;
endfor;

@ Normalize zprob to avoid overflow @
zmaxp = maxc(zprob');
zprob = exp(zprob - zmaxp + 3).*(psi');
zprob  = zprob./sumc(zprob');
/*
*******************************************
*  Generate the z's
*  z[i] is MN(1,p_i)
*  p_ik propto det(lambda_k)^{-0.5}
*     *exp(-0.5(beta_i-theta_k)'lambda_k^{-1}(beta_i-theta_k))
*     *psi_k
*  Only accept a new  psi,  if they generate
```

```
*  a ordering of psi_1 <= psi_2 <= ....
*****************************************
*/
if mmod > 1;
z      = rndzmn(zprob);
classn = sumc( z .== mint' ); @ Number of subjects assigned to the classes @
/*
*****************************************
*  DIRORD is ordered dirichlet. See /gauss/src/plbam.src
*  {psi, xgam} = dirord(alpha, xgam);
*    alpha = k x 1 vector of parameters.
* xgam = n x k matrix of ordered gamma random deviates.
*    psi = n x k matrix of ordered dirichlet probs.
*****************************************
*/
{psi, xgam } = dirord(w0+classn,xgam);
psi = psi';

@ Enforce order restrictions on classn.  @
@ Allow run of pcount violations before reorder @
rclass = rankindx(classn,1);
if not rclass == mint; @ counts violate order restrictions @
pflag = 1; @ set flag for next iteration            @
ipcount = ipcount + 1;
if ipcount >= pcount; @ let labels switch  @
pflag  = 0;
ipcount = 0;
if nmcmc <= nblow;
@ permute the assignments @
classn = sortc(classn,1);
        z      = recode(z, z.== mint', rclass);
        psi[rclass] = psi;  @ reorder psi  @
xgam[1, rclass] = xgam;
        theta[.,rclass] = theta;
        j = 1;
        lamb0  = lambda;
        lamb2 = lambdai;
       do while j <= mmod;
     lambda[iptgp[rclass[j],1]:iptgp[rclass[j],2],.] =
               lamb0[iptgp[j,1]:iptgp[j,2],.];
        lambdai[iptgp[rclass[j],1]:iptgp[rclass[j],2],.] =
               lamb2[iptgp[j,1]:iptgp[j,2],.] ;
        j = j + 1;
endo;
endif; @ if igibbs <= nblow  @
```

```
endif; @ if ipcount >= pcount @
else; @ rclass .== mint -> reset counter & flag @
ipcount = 0;
pflag   = 0;
endif; @ if not rclass == mint @

endif;    @ if mmod > 1 @

endp;

/*
************************************************************************
*  OUTPUTANAL
* Does analysis of output and save some results
************************************************************************
*/
PROC (0) = outputanal;
local  bout, sout, ebeta, sbeta, cb, rmse, fmtn1, fmtn2, fmts1, fmts2,
 knames,knames2,mmod2,a, a1,b1,iptgpt,
lambdatk,lambdamk,lambdask,fk,k,   clrate, hbclass, hbclassk,
betat, sigmat, classt, psit, thetat, lambdat, mmodt;


if flagtrue == 1; @ Know true paramaters from simulation @
load betat = betat;
load sigmat  = sigmat;
load classt = classt;
load psit = psit;
load thetat = thetat;
load lambdat = lambdat;
mmodt = maxc(classt); @ True number of mixture components @
endif;



knames = 0 $+ "Group " $+ ftocv(mint,1,0);
if flagtrue == 1;
mmodt = maxc(classt);
knames2 = 0 $+ "Group " $+ ftocv(seqa(1,1,mmodt),1,0);
endif;
@ Define formats for fancy printing @
@ Used to print a matrix of alpha & numeric variables @
let fmtn1[1,3] = "*.*lf" 10 5; @ Format for printing numeric variable  @
let fmtn2[1,3] = "*.*lf" 10  0; @ Format for numeric variable, no decimal @
```

```
let fmts1[1,3] = "-*.*s" 10 9; @ Format for alpha, left justify   @
let fmts2[1,3]  = "*.*s" 10 9; @ Format for alpha, right justify    @
format 10,5; @ Default pring format @

output file = ^outfile reset; @ outfile is the file handle for the output file  @
@ Route printed output to the defined by outfile  @
print "Results from MIXREG.GSS";
print "Hierarchical Bayesian linear regression model using MCMC.";
print "Use mixture distribution for heterogeneity";
print "Different design matrices for each subject";
print "Y_i = X_i*beta_i + epsilon_i";
print "beta_i = sum_k psi_k N(beta_i | theta_k, lambda_k)";
print "epsilon_i is N(0,sigma2*I)";
print;
print "Number of components is fixed at: " mmod;
print;
print "Ouput file: " getpath(outfile); @ File assigned to file handle outfile @
datestr(date); @ Print the current data @
print;
print;
print "------------------------------------------------------------";
print;
print "MCMC Analysis";
print;
print "Total number of MCMC iterations               = " nmcmc;
print "Number of iterations used in the analysis     = " smcmc;
print "Number in transition period                   = " nblow;
print "Number of iterations between saved iterations  = " skip-1;
print;
print "Number of subjects                 = " nsub;
print "Mean # of observations per subject  = " meanc(yrows);
print "STD  # of observations per subject  = " stdc(yrows);
print "MIN  # of observations per subject  = " minc(yrows);
print "MAX  # of observations per subject  = " maxc(yrows);
print "Total number of observations        = " ntot;
print "Number of independent variables X   = " xdim " (excluding intercept)";
print;
print "Dependent variable is " $ynames;
print;
print "Independent variables in first level equation: Y_i = X_i*beta_i + epsilon_i";
call sumstats(xnames,xdata,fmts1,fmts2,fmtn1); @ Print summary statisticcs @
print;
print "Loglikelihood form MCMC:";
print "Number of mixture components = " mmod;
print "Posterior mean               = " loglikem;
```

```
print "Posterior STD               = " loglikes;
print;
print "-------------------------------------------------------------";
print;
print "Statistics of Fit Measures for each Subject";
print "Average Predictive Correlation (Muptiple R) = " meanc(br);
print "STD of Predictive Correlations              = " stdc(br);
print "Average R-Squared                           = " meanc(br^2);
print "STD of R-Squared                            = " stdc(br^2);
print "Average Error Standard Deviation            = " meanc(estd);
print "STD of Error Standard Deviation             = " stdc(estd);
print;
print "-------------------------------------------------------------";
print "Estimation of the error STD sigma";
if flagtrue == 1;
print "True Sigma     = " sigmat;
endif;
print "MLE            = " sighat;
print "Posterior Mean = " sigmam;
print "Posterior STD  = " sigmas;
print;
print "-------------------------------------------------------------";
print "Statistics for Individual-Level Regression Coefficients";

if flagtrue == 1;
ebeta = meanc(betat');
sbeta = stdc(betat');
print "True Beta";
a = {"Variable" "Mean" "STD" };
call outitle(a,fmts1,fmts2);

bout  = xnames~ebeta~sbeta;
call outmat(bout,fmts1,fmtn1);

endif;

print "MLE Coefficients ";
a = {"Variable" "MeanMLE" "STDMLE"};
call outitle(a,fmts1,fmts2);
bout = xnames~meanc(bhat')~stdc(bhat');
call outmat(bout,fmts1,fmtn1);

print "HB Coefficients ";
a  = {"Variable"  "PostMean"  "PostSTD"};
call outitle(a,fmts1,fmts2);
```

```
ebeta = meanc(betam');
sbeta = sqrt( meanc( (betas^2)') + stdc( betam')^2);

bout  = xnames~ebeta~sbeta;
call outmat(bout,fmts1,fmtn1);

if flagtrue == 1;
print "Comparison of True Beta to Individual Level Estimates";
for i0 (1,rankx,1); i = i0;
print "Variable is " $ xnames[i];
cb  = corrx( betat[i,.]'~betam[i,.]' );
rmse  = betat[i,.] - betam[i,.];
rmse = rmse*rmse';
rmse = sqrt(rmse/nsub);
print "Correlation between true and HB  = " cb[1,2];
print "RMSE between true and HB         = " rmse;
print;
cb  = corrx( betat[i,.]'~bhat[i,.]' );
rmse  = betat[i,.] - bhat[i,.];
rmse = rmse*rmse';
rmse = sqrt(rmse/nsub);
print "Correlation between true and MLE = " cb[1,2];
print "RMSE between true and MLE        = " rmse;
print;
endfor;
endif;
print "-------------------------------------------------------------";
if mmod > 1;
print "Estimated Group Probabilities psi";

if flagtrue == 1;

a = "  "~(knames2');
call outitle(a,fmts1,fmts2);
bout = "True"~(psit');
call outmat(bout,fmts1,fmtn1);
endif;
a = "   "~(knames');
call outitle(a,fmts1,fmts2);

bout = "HB Mean"~(psim');
bout = bout|("HB STD"~(psis'));
call outmat(bout,fmts1,fmtn1);
endif;
```

```
if flagtrue == 1; @ Do Misclassification @
print "-------------------------------------------------------------";

clrate = zeros(mmod,mmodt);
hbclass = maxindc(zprobm'); @ Get index that correponds to the maximum @
for fk (1,mmodt,1); k =fk;
hbclassk = selif(hbclass, classt .== k);
if rows(hbclassk) > 0;
clrate[.,k] = sumc( hbclassk .== mint');
endif;
endfor;

print "Classification Rates:  True versus Maximum HB Posterior Probability";
a = 0 $+ "HB Groups ";
a = a|(0 $+ "True " $+ ftocv(seqa(1,1,mmodt),1,0));
a = a|"Total";
a = a';
call outitle(a,fmts1,fmts2);

a = knames|"Total";
bout = clrate~(sumc(clrate'));
bout = bout|(sumc(bout)');
bout = a~bout;
call outmat(bout,fmts1,fmtn2);
print;
endif;
print "-------------------------------------------------------------";
print "HB Estimates of Theta";

if flagtrue == 1;
print "True Theta";
a = "Variable"~(knames2');
call outitle(a,fmts1,fmts2);
bout = xnames~thetat;
call outmat(bout,fmts1,fmtn1);
endif;
print "Posterior Mean of Theta";
a  = "Variable"~(knames');
call outitle(a,fmts1,fmts2);
bout = xnames~thetam;
call outmat(bout,fmts1,fmtn1);

print "Posterior STD of Theta";
call outitle(a,fmts1,fmts2);
bout = xnames~thetas;
```

```
call outmat(bout,fmts1,fmtn1);
print "-------------------------------------------------------------";
a = "Variable"~(xnames');

print "HB Estimate of Lambda";

if flagtrue == 1;
b1 = seqa(rankx,rankx,mmodt);
a1 = 1|(1+b1[1:mmodt-1]);
iptgpt = a1~b1;
for fk (1,mmodt,1); k = fk;
lambdatk = lambdat[iptgpt[k,1]:iptgpt[k,2],.];
print "True Lambda for group " k;
call outitle(a,fmts1,fmts2);
bout = xnames~lambdatk;
call outmat(bout,fmts1,fmtn1);
endfor;
endif;
print "-------------------------------------------------------------";
for fk (1,mmod,1); k = fk;
print "Posterior Mean of Lambda for group " k;
lambdamk = lambdam[iptgp[k,1]:iptgp[k,2],.];
call outitle(a,fmts1,fmts2);
bout = xnames~lambdamk;
call outmat(bout,fmts1,fmtn1);
print "Posterior STD of Lambda for group " k;
lambdask = lambdas[iptgp[k,1]:iptgp[k,2],.];
call outitle(a,fmts1,fmts2);
bout = xnames~lambdask;
call outmat(bout,fmts1,fmtn1);
print "-------------------------------------------------------------";
endfor;
print "=============================================================";

output off;
closeall;
endp;

/*
*************************************************************
* OUTITLE
* Prints header for columns of numbers.
* INPUT
* a  = character row vector of column names
* fmts1 = format for first column
```

```
* fmts2 = format for second column
* OUTPUT
* None
************************************************************
*/
PROC (0) = outitle(a,fmt1,fmt2);
local mask, fmt, flag, ncols;
ncols = cols(a);
mask = zeros(1,ncols);
fmt = fmt1|(ones(ncols-1,1).*.fmt2);
flag = printfm(a,mask,fmt);
print;
endp;
/*
************************************************************
* OUTMAT
* Outputs a matrix:
* (Character Vector)~(Numeric matrix);
* The entries in the numeric matrix have the same format
* INPUT
* bout = matrix to be printed
* fmts = format for string
* fmtn = format for numeric matrix
* OUTPUT
* None
************************************************************
*/
PROC (0) = outmat(bout,fmts,fmtn);
local fmt,mask,flag,ncols, nrows;
ncols = cols(bout);
nrows = rows(bout);
fmt = fmts|(ones(ncols-1,1).*.fmtn);
mask = zeros(nrows,1)~ones(nrows,ncols-1);
flag = printfm(bout,mask,fmt);
print;
endp;

/*
************************************************************
* SUMSTATS
* Prints summary statistics for a data matrix
* INPUT
* names = charater vector of names
* data = data matrix
* fmts1 = format for string
```

```
* fmts2 = format for string
* fmtn = format for numbers
* OUTPUT
* None
*************************************************************
*/
PROC (0) = sumstats(names,data,fmts1,fmts2,fmtn);
local a, bout;
a = {"Variable" "Mean" "STD" "MIN" "MAX"};
call outitle(a,fmts1,fmts2);
bout = names~meanc(data)~stdc(data)~minc(data)~maxc(data);
call outmat(bout,fmts1,fmtn);
endp;
```

Chapter 7

Probit Model

7.1 Data Generation

```
/*
**********************************
*   (C) Copyright 1999, Peter Lenk. All Rights Reserved.
*  GProbit2.GSS
*-->GProbit1.GSS assumes common design matrix for all subjects
*-->GProbit2.GSS allows for different design matrices.
*       Generats data for HB PROBIT Regression Model
*       Select one of mvar+1 alternatives where the last alternative is
*       the base brand.
*
*       Y_{ij}  = X_{ij}*beta_i + epsilon_{ij}
*           for i = 1, ..., I and j = 1, ..., n_i
*       Y_{ij} is mvar vector
*       Y_{ijk} is the utility from subject i, choice set j, and alternative k
*       for     i = 1, ..., nsub
*               j = 1, ..., yrows[i]
*               k = 1, ..., mvar
*       Alternative mvar+1 is the base vector.

*
*       Select alternative k if:
*           Y_{ijk} > max( Y_{ijl}, 0} for l \= mvar+1
*       Select mvar+1 if max(Y) < 0.
*
*       beta_i is rankx vector
*       epsilon_{ij} is N(0,Sigma)
*       Identification:
*           sigma[mvar,mvar] = 1
*
*
*       X_{ij} is mvar x rankx
*           X will have brand intercepts for the first mvar-1 brands,
*           and coefficients for price and advertising.
*           To identify the model, we fix the intercept for the last brand to zero.
*
*
*       beta_i  = Theta'Z_i + delta_i
*       delta_i is N(0,Lambda)
*           Z1 is ln(income) and z2 = family size.
*
**********************************
*/
new;
```

```
flagplot = 0;                          @ 1 -> do a bunch of plots                    @
nsub    = 100;                         @ Number of subjects                          @
mvar    =  3;                          @ Y_{ij} is mvar vector. Eg: mvar+1 brands    @
                                       @ Choice mvar+1 is the base brand             @
yrows   = 10 + ceil(10*rndu(nsub,1));  @ Gives the number of observations per subject @

ntot    = sumc(yrows);                 @ total number of observations                @

rankx   = mvar + 2;         @ Brand 1, Brand 2, Brand 3, Price, Advert  @
                            @ Intercept for Brand 4 = 0                 @

rankz   =  3;               @ # rows of Z_i                                          @
                            @ intercept, ln(income), and family size                 @

@ Define some variable names for Gauss file @
@ Use string arrays                         @
a           = seqa(1,1,mvar);
brands      = 0 $+ "Brand " $+ ftocv(a,1,0);
brands2     = brands|"Brand 4";



@ ftocv converts a numeric variable to alpha and $+ is character addition       @
@ so brands is Brand 1, Brand 2, ..., Brand mvar                                 @

xyname      = brands|"Price"|"Advert"|"Choice";  @ | stacks matrices on top of each other @
zname       = {"Constant", "lnIncome",  "HH Size" };
@ To print a character string use: print $ zname; @


@ define two pointers to access xdata matrix @
@ xdata = { x_{11}, ... x_{1n_1}, x_{21}, ..., x_{2,n_2}, ..., x_{nsub,1} ... x_{nsub,n_{nsub}} } @
@ xij = xdata[lxy[i,j]:uxy[i,j],.] @
lxy     = zeros(nsub,maxc(yrows));      @ gives lower subscript @
uxy     = lxy;                          @ gives upper subscript @
s1      = 0;
for i0 (1,nsub,1); i = i0;
    for fj (1,yrows[i],1); j = fj;
        uxy[i,j]    = mvar*(s1+j);
    endfor;
    s1 = s1 + yrows[i];
endfor;
lxy     = uxy - mvar + 1;
lxy = (0-lxy).*(lxy .< 0) + lxy;            @ zero-out the negative entries. @
```

```
@ Generate error variance Sigma @

sigmat  = { .2  .1  -.1,
            .1  .3  -.05,
            -.1   -.05  1
};
sigt12  = chol(sigmat);


@ Generate error variance Lambda @

@           b1      b2      b3          Price   Advert      @
lambdat = {
            1       .3      -.1         0   0   ,       @ b1        @
            .3      .8      -.05        0   0   ,       @ b2        @
            -.1     -.05    .5          0   0   ,       @ b3        @
            0       0       0           .2  .1  ,       @ price     @
            0       0       0           .1  .5          @ advert    @
        };
lambdat = 0.01*lambdat;
lbd12   = chol(lambdat);


@ generate Z variables  @
@Z1 is ln(income) @
m       = ln(40000);
s       = (ln(120000) - m)/1.96;
z1      = m + s*rndn(nsub,1);
@ Mean center z1@
z1      = z1 - meanc(z1);

@Z2 is family size @
z2      = floor(rndnab(nsub,1,3,4,1,10));
@rndnab is my truncated normal(rows, cols, mean, std, a, b)@
zdata   = z1~(z2-meanc(z2));
zdata   = ones(nsub,1)~zdata;



@ Generate theta @
```

```
@       B1 - B4     B2-B4   B3-B4   Price   Advertising                @

thetat  = {
            1       .5       0      -2      1,      @ Intercept         @
            1       .5       -.3    1       -.5,    @ Ln(Income)        @
            -.3     -.2     0       -.3     .5      @ Family Size       @
        };

@ Generate partworths beta @
betat   = zdata*thetat  + rndn(nsub,rankx)*lbd12;


@ generate X & Y data @
ydata   = zeros(ntot*mvar,1);               @ 0/1 choice    @
ydatat  = ydata;                            @ true utilities of Brand j - Brand mvar+1 @
xdata   = zeros(ntot*mvar,rankx);
ipick   = zeros(mvar+1,1);      @ keep track of the number of choices @
for i0 (1,nsub,1); i = i0;
    for fj (1,yrows[i],1); j = fj;
        @ Do subject i, purchase j @
        @ xij = brands, price, advertising @
        xij = eye(mvar);                        @ Brand Intercepts @
        @ Generate Prices                                           @
        @          Regular Price            Price Promotion         @
        p1      = 5.5 + .1*rndn(1,1) - (rndu(1,1) < .4)*.3;
        p2      = 5.5 + .1*rndn(1,1) - (rndu(1,1) < .4)*.3;
        p3      = 5.2 + .1*rndn(1,1) - (rndu(1,1) < .2)*.2;
        p4      = 5.0 + .05*rndn(1,1);
        price   = p1-p4|p2-p4|p3-p4;
        xij     = xij~price;
        @ Do advertising            @
        @ Brand 1 heavily advertises, followed by the other four        @
        a1      = rndu(1,1) < .4;
        a2      = rndu(1,1) < .4;
        a3      = rndu(1,1) < .2;
        a4      = rndu(1,1) < .1;
        advert  = a1-a4|a2-a4|a3-a4;
        xij     = xij~advert;

        @ Generate utility for the four brands @
        bi          = betat[i,.]';
        yij         = xij*bi + sigt12'rndn(mvar,1);
        choice      = zeros(mvar,1);
        ib          = maxindc(yij|0);
        if ib <= mvar;
```

```
            choice[ib]  = 1;
        endif;
        ipick[ib]   = ipick[ib] + 1;


        @ Save it in the data matrix            @
        ydata[lxy[i,j]:uxy[i,j],.]      = choice;
        ydatat[lxy[i,j]:uxy[i,j],.]     = yij;
        xdata[lxy[i,j]:uxy[i,j],.]      = xij;
        @ rows of ydata2 are purchase occassions and columns are brands @
        @ ydata2 will be used for computing summary statiscs for utilities @
        if i == 1 and j == 1;
            ydata2 = yij';
        else;
            ydata2 = ydata2|(yij');
        endif;
    endfor;
endfor;
xydata          = xdata~ydata;

/*
****************************************
* Output to a Gauss file.
* Gauss files are faster to read, and they assign variable names to
* the columns.  Also, Gauss has a number of special commands that operate
* on Gauss files, such as file merges, variable recoding, and summary statistics.
*
* First, create a Gauss file.  f1 is the file handle.
* The Gauss file will be called "XPDATA."
* The column will be named by the strings in the character array xyname.
* ^xyname means use the names in the character string.
* 0, 8 gives double precision real numbers.
*************************************
*/

create f1 = xpdata with ^xyname, 0, 8;

/*
****************************************
* Next read data into the Gauss file by using the writer command.
* f1 is the file handle defined in previous command.
* xydata is the data matrix that we just created.
* writer returns the number of rows read to f1.
* If it is not rows(xydata), something bad happended.
****************************************
```

```
*/

if writer(f1,xydata) /= rows(xydata);
        errorlog "Conversion of XYDATA to Gauss File did not work";
endif;
closeall f1;
@ We do not need f1 anymore, so close it up.                                @
@ Do the same for zdata                                                     @
create f1 = zdata with ^zname, 0, 8;
if writer(f1,zdata) /= rows(zdata);
    errorlog "Conversion of ZDATA to Gauss File did not work";
endif;
closeall f1;

save yrows      = yrows;
save lxy        = lxy;
save uxy        = uxy;

save ydatat     = ydatat;
save sigmat     = sigmat;
save betat      = betat;
save thetat     = thetat;
save lambdat    = lambdat;

@ Define formats for fancy printing @
@ Used to print a matrix of alpha & numeric variables @
let fmt1a[1,3]  = "*.*lf" 10 5;         @ Format for printing numeric variable      @
let fmtsb[1,3]  = "*.*s" 8 8;           @ Format for printing character variable    @
mask    = zeros(mvar+1,1)~ones(mvar+1,1);   @ 0 for alpha, and 1 for numeric            @
fmt1    = fmtsb|fmt1a;                  @ Format for columns of output              @
bout    = brands2~(ipick/ntot*100);
print "    Brand Market Shares";
print " Brand    Market Share (%)";
flag        = printfm(bout,mask,fmt1);  @ Formated print.                           @
print;
mask    = zeros(mvar,1)~ones(mvar,4);
fmt2    = fmtsb|fmt1a|fmt1a|fmt1a|fmt1a;
bout    = brands~meanc(ydata2)~stdc(ydata2)~minc(ydata2)~maxc(ydata2);
print "    Summary Statistics for Brand by Purchase Occasion Utilities";
print " Brand     Mean      STD       MIN       MAX";
flag    = printfm(bout,mask,fmt2);

if flagplot == 1;
_plctrl = -1;           @ use symbols only in plots @
@ plot beta versus ln(income) and family size @
```

```
for fj (mvar,rankx,1); j = fj;
    atitle = "Partworth for " $+ xyname[j] $+ " versus " $+ zname[2];
    title(atitle);
    xy(zdata[.,2], betat[.,j]);
    atitle = "Partworth for " $+ xyname[j] $+ " versus " $+ zname[3];
    title(atitle);
    xy(zdata[.,3], betat[.,j]);
    wait;                          @ Hit any key to continue @
endfor;

@ plot y versus price for the mvar brands @
for fj (1,mvar-1,1); j = fj;
    @ create a vector to select brands from data matrices @
    b       = zeros(mvar,1); b[j] = 1;
    bd      = ones(ntot,1).*.b;  @ .*. is Kronecker product. bd is a vector of 0 & 1 @
    xj      = selif(xdata, bd);  @ selif extracts the rows where bd = 1 @
    yj      = selif(ydatat, bd);
    atitle = "Utility for " $+ brands[j] $+ " versus " $+ xyname[mvar];
    title(atitle);
    xy(xj[.,mvar], yj);
    atitle = "Utility for " $+ brands[j] $+ " versus " $+ xyname[mvar+1];
    title(atitle);
    xy(xj[.,mvar+1], yj);
    wait;
endfor;

graphset;           @ Set plots back to default values @

endif;
end;
```

7.2 Bayesian Analysis

```
/*
************************************
*   (C) Copyright 1999, Peter Lenk. All Rights Reserved.
*   PROBIT2.GSS
*   HB Probit Regression Model.
*-->PROBIT1.GSS has common design matrix for all subjects
*-->PROBIT2.GSS allows different design matrices.
*
*   Uses McCulloch & Rossi's method for handling identification.
*
*       Select one of mvar+1 alternatives.
*
*       Y_{ij}  = X_{ij}*beta_i + epsilon_{ij}
*           for i = 1, ..., I and j = 1, ..., n_i
*       Y_{ij} is mvar vector
*       Y_{ijk} is the utility from subject i, choice set j, and alternative k
*       for     i = 1, ..., nsub
*               j = 1, ..., yrows[i]
*               k = 1, ..., mvar
*
*       Alternative mvar+1 is the base vector.
*
*
*       Select alternative k if:
*           Y_{ijk} > max( Y_{ijl} } for 1 \= k < mvar+1.
*       Select base brand if max(Y) < 0.
*
*       Observe the choices, not the utilities Y_{ij}.
*       Pick_{ij} is a mvar vector of 0/1.
*
*
*       beta_i is rankx vector
*       epsilon_{ij} is N(0,Sigma)
*       Sigma is full
*       Divide by last element after MCMC to identify coefficients.
*
*
*       X_{ij} is mvar x rankx
*           X will have brand intercepts,
*           and coefficients for price and advertising relative to the base brand.
*
*
*       beta_i  = Theta'Z_i + delta_i
```

```
*       delta_i is N(0,Lambda)
*           Z1 is ln(income) and z2 = family size.
*   PRIORS
*       Sigma  is Inverted Wishart(sf0,sg0)
*       Theta is maxtrix normal(u0,v0).
*       That is, vec(Theta) is N(vec(u0),v0).
*       vec(theta) stacks the columns of theta.
*       Lambda is Inverted Wishart(f0, g0)
************************************
*/
new;
outfile     = "results1.dat";   @ Specify output file for saving results                    @
                                @ outfile is a string variable that contains a file name    @
inxy        = "xpdata";         @ Name of Gauss file with X, Choice data                    @
inz         = "zdata";          @ Name of Gauss file with Z data                            @
flagtrue    = 1;                @ 1 -> knows true parameters from simulation                @

/*
************************************
*   Initialize parameters for MCMC
************************************
*/
smcmc       = 200;      @ number of iterations to save for analysis                 @
skip        = 1;        @ Save every skip iterations                                @
nblow       = 100;      @ Initial transition iterations                             @
nmcmc       = nblow + skip*smcmc;   @ total number of iterations                    @
nygen       = 1;        @ Do nygen generations of Y for each MCMC iteration.        @

/*
************************************
*   Get data
************************************
*/

@ Get dimensions and pointers @
load yrows  = yrows;    @ Number of observations per subject    @
load lxy    = lxy;      @ xij = xdata[lxy[i,j]:uxy[i,j],.]      @
load uxy    = uxy;

nsub    = rows(yrows);                  @ number of subjects                    @
mvar    = uxy[1,2] - uxy[1,1];          @ Y_{ij} is a mvar vector.              @
ntot    = sumc(yrows);

@ Input Gauss files @
open f1     = ^inxy;                    @ Get Gauss file for X, Y data              @
```

```
                                        @ Opens Gauss file & assigns file handle f1 @
xpdata      = readr(f1,rowsf(f1));      @ "p" for picks                              @
            @ readr reads in Gauss file with file handle f1.           @
            @ rowsf(f1) returns the number of rows in the Gauss file.  @
            @ readr reads rowsf(f1) rows, which is the entir dataset.  @
ci          = close(f1);
xpnames     = setvars(inxy);            @ Get the variable names that accompnay X, Y data @
ynames      = xpnames[1:mvar];          @ Use names of interceps for names of components of Y @
ynames2     = ynames|" Base ";
rankxp      = cols(xpdata);
rankx       = rankxp - 1;                   @ # of X variables (includes intercept) @
xnames      = xpnames[1:rankx];
@ Last row of xpdata is the choice vector. @

open f1     = ^inz;
zdata       = readr(f1,rowsf(f1));      @ First column of zdata is a vector of ones     @
ci          = close(f1);
znames      = setvars(inz);
rankz       = cols(zdata);                  @ # of Z variables (includes intercept) @
thdim       = rankx*rankz;                  @ dimension of vec(theta) @

@ Compute some sufficient statistics @
ztz     = zdata'zdata;

/*
************************************
*   Initialize Priors
************************************
*/


@ Prior for theta is N(u0,v0) @

u0      = zeros(thdim,1);
v0      = 100*eye(thdim);       @ thdim = rankx*rankz    @
v0i     = invpd(v0);            @ used in updating theta @
v0iu0   = v0i*u0;               @ used in updating theta @

@ Prior for sigma is IW(sf0, gs0) @
sf0     = mvar+2; sfn = sf0 + ntot;
sg0i    = eye(mvar);



@ Lambda^{-1} is W_rankx(f0,g0 ) @
```

```
@ f0 = prior df, g0 = prior scale matrix @
f0      = rankx+2;  f0n = f0 + nsub;
g0i     = eye(rankx);   @ g0^{-1} @

/*
************************************
*   Initialize MCMC
************************************
*/
ydata   = xpdata[.,rankxp];     @ latent y variables        @
beta    = zeros(nsub,rankx);
sigma   = eye(mvar);
sigmai  = invpd(sigma);

theta   = zeros(rankz,rankx);
lambda  = eye(rankx);
lambdai = invpd(lambda);

@ Define data structures for saving iterates & computing posterior means & std @
betam   = zeros(nsub,rankx);    @ posterior mean of beta        @
betas   = zeros(nsub,rankx);    @ posterior std of beta         @
c       = mvar*(mvar+1)/2;
sigmag  = zeros(smcmc,c);       @ save iterations for sigma     @
thetag  = zeros(smcmc,thdim);
c       = rankx*(rankx+1)/2;
lambdag = zeros(smcmc,c);       @ save iterations for lambda    @
ydatam  = zeros(mvar*ntot,1);   @ posterior mean utilities      @
ydatas  = ydatam;               @ posterior std utilities       @



/*
************************************
*   Do MCMC
************************************
*/
etime = hsec;
@ Do the initial transition period @
for i1 (1,nblow,1); imcmc = i1;
    call getprobit;
    if imcmc == 100*floor(imcmc/100);
        dtime = (hsec -etime)/(60*100);
        print "TP Iteration = " imcmc " D.time = " dtime;
        etime = hsec;
    endif;
```

```
endfor;

etime = hsec;
for i1 (1,smcmc,1); imcmc = i1;     @ Save smcmc iterations             @
    for i2 (1,skip,1); jmcmc = i2;      @ Save every skip iterations    @
        call getprobit;
    endfor;
    if imcmc == (100/skip)*floor(skip*imcmc/100);
        dtime   = (hsec - etime)/(60*100);
        tit     = nblow + skip*imcmc;
        print "Iteration = " tit " D.time = " dtime;
        etime   = hsec;
    endif;
    @ When saving parameter, divide by sqrt(sigma[mvar,mvar]) or sigma[mvar,mvar] @
    sqrtsiglast     = sqrt(sigma[mvar,mvar]);
    sigmag[imcmc,.] = vech(sigma/sigma[mvar,mvar])';
                                  @ vech({1 2 3, 4 5 6, 7 8 9}) = {1, 4 5, 7 8 9} @
                                  @ xpnd is the inverse operator of vech          @
    thetag[imcmc,.] = vecr(theta/sqrtsiglast)';
    betam           = betam     + beta/sqrtsiglast;
    betas           = betas     + (beta/sqrtsiglast)^2;
    lambdag[imcmc,.]= vech(lambda/sigma[mvar,mvar])';
    ydatam          = ydatam + ydata/sqrtsiglast;
    ydatas          = ydatas + (ydata/sqrtsiglast)^2;
endfor;


/*
**********************************
*   Compute Posterior Means and STD
**********************************
*/


ydatam      = ydatam/smcmc;
betam       = betam/smcmc;
thetam      = reshape(meanc(thetag),rankz,rankx);
sigmam      = xpnd(meanc(sigmag));      @ xpnd reconstructs symmetric matrix @
lambdam     = xpnd(meanc(lambdag));

ydatas      = sqrt( abs(ydatas  - smcmc*ydatam^2)/smcmc);
betas       = sqrt( abs(betas   - smcmc*betam^2)/smcmc);
thetas      = reshape(stdc(thetag),rankz,rankx);
sigmas      = xpnd(stdc(sigmag));
lambdas     = xpnd(stdc(lambdag));
```

```
if flagtrue == 1;           @ Did a simulation, so we have the true utilities.  @
    @ Get true parameters if simulation @

    load ydatat     = ydatat;
    load betat      = betat;
    load sigmat     = sigmat;
    load thetat     = thetat;
    load lambdat    = lambdat;



    @ Pick out each dimension of Y_{ij} and compute fit statistics @
    multir  = zeros(mvar,1);
    rsquare = zeros(mvar,1);
    stderr  = zeros(mvar,1);
    for fm (1,mvar,1); m = fm;
        b       = zeros(mvar,1); b[m] = 1;
        bn      = ones(ntot,1).*.b;
        ym      = selif(ydatat,bn);
        yhatm   = selif(ydatam, bn);
        cm      = corrx(ym~yhatm);
        multir[m]   = cm[1,2];
        rsquare[m]  = cm[1,2]^2;
        resid       = ym - yhatm;
        stderr[m]   = sqrt(resid'resid/ntot);
    endfor;
endif;


/*
**********************************
*   Do some output
**********************************
*/
call outputanal;


@ Plot saved iterations against iterations number @
t   = seqa(nblow+skip,skip,smcmc);      @ saved iteration number @
title("Latent Error Cov vs Iteration");
xy(t,sigmag);
title("Theta vs Iteration");
xy(t,thetag);
title("Lambda vs Iteration");
```

```
xy(t,lambdag);
graphset;

end;

/*
**********************************
* GETPROBIT
*   Does one iteration of the HB regression model.
*   INPUT
*       Global Variables
*   OUTPUT
*       Global Variables
**********************************
*/
PROC (0) = getprobit;
local zbl, bi, vibn, vibn12, ebin, yhat, sse, sn, resid, gni, gn, gn12, w, sum1, sum2,
i0, i, fj, j, xij, yij, sgni, sgn, sgn12, muij, cij, ic,
v, sig11, sig11i, smigni, signi;


    /*
    ************************************
    * Compute quantities used in conditional normal distribution.
    * Need to run cndcov(sigma) before generating the random utilities.
    ************************************
    */
    {smigni, signi} = cndcov(sigma);
    /*
    ************************************
    * smigni is a mvar x (mvar-1) matrix and
    * used in the conditional mean of Y_{i} given Y_{not i}:
    * smigni[i,.] = sigma_{i, not i}*sigma_{not i, not i)^{-1}
    * signi is a mvar matrix and
    * signi[i] = STD(Y_{i}| Y_{not i})
    *          = sqrt(sigma_{ii} - sigma_{i,not i}*sigma_{not i, not i}^{-1} sigma_{not i,i})
    ************************************
    */



    /*
    ************************************
    * Generate Y_{ij}, the utility.
    *
```

```
* If alternative k (k = 1, .., mvar) was selected, then
* Y_{ij} is N(X_{ij}*beta_i, Sigma) and Y_{ij}[k] >= max(Y_{ij})
**********************************
*/
@ Do multiple loops of generating the Utilities for each MCMC Iteration @

for i0 (1, nsub, 1); i = i0;
    for fj (1,yrows[i],1); j = fj;
        xij     = xpdata[lxy[i,j]:uxy[i,j],1:rankx];
        cij     = xpdata[lxy[i,j]:uxy[i,j],rankxp];     @ Choice vector     @
        yij     = ydata[lxy[i,j]:uxy[i,j]];
        muij    = xij*(beta[i,.]');                     @ Mean for y_{ij}   @

        if maxc(cij) == 0;      @ selected base brand mvar + 1@
            ic = mvar+1;
        else;               @ selected one of brand 1 to mvar @
            ic = maxindc(cij);
        endif;
        yij = rndnigtj(yij, ic, muij, smigni, signi, nygen);
        ydata[lxy[i,j]:uxy[i,j]] = yij;                 @ store the utility            @
    endfor;
endfor;

/*
**********************************
* Generate beta
* beta_i is N(mbin, vbn)
* vbn   = ( sum_{j=1}^{n_i} X_{ij}'Sigma^{-1} X_{ij} + Lambda^{-1} }^{-1}
* mbin  = vbn*( sum_{j=1}^{n_i} X_{ij}'Sigma^{-1}Y_{ij} + Lambda^{-1}*Theta*Z_i)
**********************************
*/
zbl     = zdata*theta*lambdai;
for i0 (1, nsub,1); i = i0;
    sum1    = 0;
    sum2    = 0;
    for fj (1,yrows[i],1); j = fj;
        xij     = xpdata[lxy[i,j]:uxy[i,j],1:rankx];
        yij     = ydata[lxy[i,j]:uxy[i,j]];
        sum1    = sum1 + xij'sigmai*xij;
        sum2    = sum2 + xij'sigmai*yij;
    endfor;
    vibn        = sum1 + lambdai;
    vibn12      = chol(vibn);
    ebin        = sum2 + zbl[i,.]';
    bi          = cholsol(ebin + vibn12'rndn(rankx,1), vibn12);
```

```
        beta[i,.]   = bi';

    endfor;

    /*
    ********************************
    * Generate sigma
    ********************************
    */
    @ Compute SSE       @
    sse = zeros(mvar,mvar);
    for i0 (1, nsub,1); i = i0;
        for fj (1,yrows[i],1); j = fj;
            xij     = xpdata[lxy[i,j]:uxy[i,j],1:rankx];
            yij     = ydata[lxy[i,j]:uxy[i,j]];
            resid   = yij - xij*(beta[i,.]');
            sse     = sse + resid*resid';
        endfor;
    endfor;
    sgni                = sg0i + sse;
    sgn                 = invpd(sgni);
    {sigmai, sigma}    = wishart(mvar,sfn,sgn);



    /*
    ********************************
    * Generate Theta and Lambda from multivariate model:
    *   B = Z*Theta + N(0,Lambda)
    ****************************
    */
    {theta, lambda, lambdai} =
    getmulreg(beta,zdata,ztz,theta,lambda,lambdai,v0i,v0iu0,f0n,g0i);

endp;


/*
********************************
* GETMULREG
*   Generate multivariate regression parameters.
*       Yd      = Xd*parmat + epsilon
*
*   INPUT
*       yd      = dependent variables
```

```
*       xd      = independet variables
*       xdtxd       = xd'xd
*
*       parmat  = current value of coefficient matrix
*       var     = current value of covariance matrix
*       vari        = its inverse
*       v0i     = prior precisions for bmat
*       v0iu0       = prior precision*prior mean for bmat
*       f0n     = posterior df for sigma
*       g0i     = prior scaling matrix inverse for sigma

*
*   OUTPUT
*       parmat  = updated rankx x mvar coefficient matrix
*       var     = updated variance
*       vari        = updated inverse of sigma
*
*   Calling Statement:
{parmat, var, vari} = getmulreg(yd,xd,xdtxd,parmat,var,vari,v0i,v0iu0,f0n,g0i);
**********************************
*/
PROC (3) = getmulreg(yd,xd,xdtxd,parmat,var,vari,v0i,v0iu0,f0n,g0i);
local vb12, ubn, par, pdim, resid, gni, gn, rp, cp;

    rp      = rows(parmat);
    cp      = cols(parmat);
    pdim    = rp*cp;

    /*
    **********************************
    * Generate parmat from N_{rp x cp}(M,v)
    * par = vecr(parmat)
    * par is N(u,V) whee u = vec(M');
    *   V = (Xd'Xd.*.Var^{-1} + V_0^{-1})^{-1}
    *   u = V*( (Xd'.*.Var^{-1})*vec(Yd') + V_0^{-1}u_0 )
    ******************************
    */

    vb12    = chol(xdtxd.*.vari + v0i);
    ubn     = ( (xd').*.vari )*vecr(yd) + v0iu0;
    par     = cholsol(ubn + vb12'rndn(pdim,1), vb12);
    parmat  = reshape(par,rp,cp);

    /*
    ******************************
```

```
    * Generate Var
    *   Var^{-1} is Wishart df = f0n, scale matrix = gn
    *******************************
    */
    resid           = yd - xd*parmat;
    gni             = g0i + resid'resid;
    gn              = invpd(gni);
    {vari, var}     = wishart(cp,f0n,gn);

retp(parmat,var,vari);
endp;


/*
***********************************
* CNDCOV
* Generates some quantities used in computing conditional normal distribtions.
* INPUT:
*   sigma   = Cov(Y)
*
* OUPUT:
*   smigni  = used in E(Y_i|Y_{not i}).
*   signi   = STD(Y_i |Y_{not i}).
*
* smigni is a mvar x mvar-1 matrix and
* smigni[i,.] = sigma_{i, not i}*sigma_{not i, not i)^{-1}
* signi is a mvar vector and
* signi[i] =
* sqrt(sigma_{ii} - sigma_{i,not i}*sigma_{not i, not i}^{-1} sigma_{not i,i})
***********************************
*/
PROC (2) = cndcov(sigma);
local smigni, signi, sqmvar, ei, einot, s2noti, sinoti, s2notii, i0, i;

sqmvar  = seqa(1,1,mvar);
smigni  = zeros(mvar,mvar-1);       @ Sigma_{ij}*Sigma_{jj}^{-1}    @
@ smuigj is used in computing mean of i given not i                 @
signi   = zeros(mvar,1);            @ sigma_{ii} - sigma_{ij}*Sigma_{jj}^{-1}*sigma_{ji} @
@ signi is STD of Y[i] given not i @
for i0 (1,mvar,1); i = i0;
    ei      = sqmvar .== i;         @ ei[i] = 1, and ei[j] = 0 for j /= i               @
    einot   = 1 - ei;               @ einot[i] = 0, and einot[j] = 1 for j /= i         @
    @ selif(x,e) selects rows of x with e[j] = 1 @
    s2noti  = selif( selif(sigma,einot)',einot)';
    @ s2noti = sigma[j,k] with j \= i and k \= i @
```

```
    sinoti  = selif( selif(sigma',ei)',einot);
    @ sinoti is the ith column of sigma with the ith row removed @

    s2notii = invpd(s2noti);
    smigni[i,.] = sinoti's2notii;
    signi[i]    = sqrt(sigma[i,i] - smigni[i,.]*sinoti);
endfor;
retp(smigni, signi);
endp;

/*
************************************
* RNDNIGTJ.GSS
*   Generate a vector normal where the ith elememt is greater than
*   the rest.
*   INPUT
*       Y       = current value of  Y form MCMC Iterations
*       IPICK   = which component is greater than the rest, and zero
*       mu      = mean vector
*       smigni  = used in computing mean of y[i] given not y[i].
*       signi   = std of y[i] given not y[i].
        nygen   = number of generations before returning random utility.
*   OUTPUT
*       Y       = mvar vector where Y[i] > max( Y[j]  for j not equal to i )
****************************************
*/

PROC rndnigtj(y,ipick,mu, smigni, signi, nygen);
local mvar, sqmvar,ei,einot,munoti,ynoti,
mui, sii, ybot,ytop,yi, i, fori, j, fj ;

mvar    = rows(mu);                 @ Dimension of problem                  @
sqmvar  = seqa(1,1,mvar);

for fj (1, nygen, 1); j = fj;       @ Do muliple loops of generating Y                  @
for fori (1,mvar,1); i = fori;
    ei      = sqmvar .== i;         @ ei[i] = 1, and ei[j] = 0 for j /= i               @
    einot   = 1 - ei;               @ einot[i] = 0, and einot[j] = 1 for j /= i         @

    @ selif(x,e) selects rows of x with e[j] = 1 @
    ynoti   = selif(y,einot);       @ mvar - 1 vector that does not have y[i]           @
    munoti  = selif(mu,einot);      @ munoti is mvar-1 vector that does not have mu[i]  @

    mui     = mu[i] + smigni[i,.]*(ynoti - munoti);
    sii     = signi[i];     @ STD of y[i] given rest @
```

```
    if i == ipick;          @ i is the selected choice, yi > max(yj,0)@
        ybot        = maxc(ynoti|0);
        y[i]        = rndtnb(mui,sii,ybot);
        @ generates a random normal, trucnated below (in plrand.src) @
        @ rndtnb(mu,std,bot): mu = vector or means, std = vector of STD, bot = lower limit @
    else;                   @ i was not selected    @
        if ipick <= mvar;
            ytop        = y[ipick];             @ Maximum element @
        else;       @ Base brand mvar+1 selectd, so y < 0 @
            ytop        = 0;
        endif;
        y[i]        = rndtna(mui,sii,ytop);
        @ generates a random normal, truncated above (in plrand.src) @
        @ rndtna(mu,std,top): mu = vector of mean, std = vector of STD, top = upper limit @
    endif;
endfor;     @ End loop over generating one Y vector     @
endfor;     @ End loop over generating nygen Y vectors  @
retp(y);
endp;



/*
************************************************
*  OUTPUTANAL
*   Does analysis of output and save some results
************************************************
*/
PROC (0) = outputanal;
format 10,5;
local  bout, sout, ebeta, sbeta, cb, rmse, fmts1,fmts2, fmtn1, fmtn2, a,b, flag, i0, i;



let fmtn1[1,3]  = "*.*lf" 10 5;         @ Format for printing numeric variable      @
let fmtn2[1,3]  = "*.*lf" 10  0;        @ Format for numeric variable, no decimal   @

let fmts1[1,3]  = "-*.*s" 10 9;         @ Format for alpha, left justify            @
let fmts2[1,3]  = "*.*s" 10 9;          @ Format for alpha, right justify           @
format 10,5;                            @ Default pring format                      @


output file = ^outfile reset;   @ outfile is the file handle for the output file    @
                                @ Route printed output to the defined by outfile    @
```

```
print "Results from PROBIT1.GSS";
print "Hierarchical Bayes Multivariate Probit";
print;
print "Select one of mvar+1 alternatives.";
print "Alternative mvar+1 is the base alternative.";
print "Observe choice k if Y_{ij}[k] >= max(Y_{ij},0).";
print "Choose base if all the y's < 0.";
print;
print "Latent Utility Model:";
print "Y_{ij} = X_{ij}*beta_i + epsilon_{ij}";
print "Y_{ij} is a mvar vector";
print "epsilon_{ij} is N(0,Sigma).";
print "Assumes that Sigma[1,1] = 1";
print;
print "beta_i = Theta'z_i + delta_i";
print "delta_i is N(0, Lambda)";
print;
print "Ouput file: " getpath(outfile);      @ File assigned to file handle outfile @
datestr(date);                               @ Print the current data                @
print;
print;
print "-----------------------------------------";
print;
print "MCMC Analysis";
print;
print "Total number of MCMC iterations              = " nmcmc;
print "Number of iterations used in the analysis    = " smcmc;
print "Number in transition period                  = " nblow;
print "Number of iterations between saved iterations = " skip-1;
print;
print "-----------------------------------------";
print "Number of subjects                 = " nsub;
print "Number of observations per subject:";
print "     Average  = " meanc(yrows);
print "     STD      = " stdc(yrows);
print "     MIN      = " minc(yrows);
print "     MAX      = " maxc(yrows);
print;
print "Total number of Y observations     = " ntot;
print "Dimension of Y_{ij}                = " mvar;
print "Number of alternatives             = " mvar+1;
print "Number of dependent variables X    = " rankx " (including intercept)";
print "Number of dependent variables Z    = " rankz " (including intercept)";
print;
```

```
print "Dependent variables are " $xpnames[rankxp];
print "     Summary Statistics for Choices";
call sumstats(ynames,reshape(xpdata[.,rankxp],ntot,mvar),fmts1,fmts2,fmtn1);
print;
print "First level equation for Latent Utilities";
print "Y_{ij} = X_{ij}*beta_i + epsilon_{ij}";
print;
print "     Summary Statistics for X";
call sumstats(xnames,xpdata[.,1:rankx],fmts1,fmts2,fmtn1);
print;
print "Second level equation:";
print "beta_i = Theta'z_i + delta_i";
print;
print "     Summary Statistis for Z:";
call sumstats(znames,zdata,fmts1,fmts2,fmtn1);
print;


if flagtrue == 1;       @ Print some fit for utilities @
    print "-----------------------------------------";
    print "Fit between true and estimated utilities for each dimension.";
    sout    = {"Variable" "Multi-R" "R-Sqr" "ErrorSTD"};
    call outitle(sout,fmts1,fmts2);
    bout    = ynames~multir~rsquare~stderr;
    call outmat(bout,fmts1,fmtn1);
endif;
print "-----------------------------------------";
print;
print "Statistics for Individual-Level Regression Coefficients";
if flagtrue == 1;
    ebeta   = meanc(betat);
    sbeta   = stdc(betat);
    print "True Beta";
    sout    = {"Variable" "Mean" "STD"};
    call outitle(sout,fmts1,fmts2);
    bout = xnames~ebeta~sbeta;
    call outmat(bout,fmts1,fmtn1);
endif;

print "HB Estimates of Beta";
sout    = {"Variable" "PostMean" "PostSTD" };
call outitle(sout,fmts1,fmts2);
ebeta   = meanc(betam);
```

```
sbeta   = sqrt( meanc( (betas^2)) + stdc( betam)^2);
bout = xnames~ebeta~sbeta;
call outmat(bout,fmts1,fmtn1);
print;
print "-----------------------------------------";

if flagtrue == 1;
    print "Comparison of True Beta to Individual Level Estimates";
    for i0 (1,rankx,1); i = i0;
        print "Variable " $ xnames[i];
        cb      = corrx( betat[.,i]~betam[.,i] );
        rmse    = betat[.,i] - betam[.,i];
        rmse    = rmse'rmse;
        rmse    = sqrt(rmse/nsub);
        print "Correlation between true and HB  = " cb[1,2];
        print "RMSE between true and HB         = " rmse;
        print;
    endfor;
endif;
print "-----------------------------------------";
print "Estimation of the error covariance Sigam";
sout    = "   "~(ynames');

if flagtrue == 1;
    print "True Sigma";
    call outitle(sout,fmts1,fmts2);
    bout    = ynames~sigmat;
    call outmat(bout,fmts1,fmtn1);
    print;
endif;
print;
print "Posterior Mean of Sigma";
call outitle(sout,fmts1,fmts2);
bout    = ynames~sigmam;
call outmat(bout,fmts1,fmtn1);
print;
print "Posterior STD  of Sigma";
call outitle(sout,fmts1,fmts2);
bout    = ynames~sigmas;
call outmat(bout,fmts1,fmtn1);
print;
print "-----------------------------------------";
print;
print "HB Estimates of Theta";
sout    = "  "~(xnames');
```

```
if flagtrue == 1;
    print "True Theta";
    call outitle(sout,fmts1,fmts2);
    bout = znames~thetat;
    call outmat(bout,fmts1,fmtn1);
    print;
endif;
print "Posterior Mean of Theta";
print outitle(sout,fmts1,fmts2);
bout = znames~thetam;
call outmat(bout,fmts1,fmtn1);
print;
print "Posterior STD of Theta";
call outitle(sout,fmts1,fmts2);
bout    = znames~thetas;
call outmat(bout,fmts1,fmtn1);
print;
print "-----------------------------------------";
print;
sout = "  "~(xnames');
print "HB Estimate of Lambda";
if flagtrue == 1;
    print "True Lambda";
    call outitle(sout,fmts1,fmts2);
    bout = xnames~lambdat;
    call outmat(bout,fmts1,fmtn1);
    print;
endif;
print "Posterior Mean of Lambda";
call outitle(sout,fmts1,fmts2);
bout = xnames~lambdam;
call outmat(bout,fmts1,fmtn1);
print;
print "Posterior STD of Lambda";
call outitle(sout,fmts1,fmts2);
bout = xnames~lambdas;
call outmat(bout,fmts1,fmtn1);
print;
print "=========================================";



output off;
closeall;
endp;
```

```
/*
****************************************
*  TR(X)
*  Compute the trace of X. Functional call (FN).
****************************************
*/
fn tr(x) = sumc(diag(x));



/*
************************************************
* OUTITLE
*   Prints header for columns of numbers.
*   INPUT
*       a   = character row vector of column names
*       fmts1   = format for first column
*       fmts2   = format for second column
*   OUTPUT
*       None
*******************************************
*/
PROC (0) = outitle(a,fmt1,fmt2);
local mask, fmt, flag, ncols;
ncols   = cols(a);
mask    = zeros(1,ncols);
fmt     = fmt1|(ones(ncols-1,1).*.fmt2);
flag    = printfm(a,mask,fmt);
print;
endp;
/*
*********************************************
* OUTMAT
*   Outputs a matrix:
*   (Character Vector)~(Numeric matrix);
*   The entries in the numeric matrix have the same format
*   INPUT
*       bout        = matrix to be printed
*       fmts        = format for string
*       fmtn        = format for numeric matrix
*   OUTPUT
*       None
*********************************************
```

```
*/
PROC (0) = outmat(bout,fmts,fmtn);
local fmt,mask,flag,ncols, nrows;
ncols       = cols(bout);
nrows       = rows(bout);
fmt         = fmts|(ones(ncols-1,1).*.fmtn);
mask        = zeros(nrows,1)~ones(nrows,ncols-1);
flag        = printfm(bout,mask,fmt);
print;
endp;

/*
************************************************
* SUMSTATS
*   Prints summary statistics for a data matrix
*   INPUT
*       names       = charater vector of names
*       data        = data matrix
*       fmts1       = format for string
*       fmts2       = format for string
*       fmtn        = format for numbers
*   OUTPUT
*       None
***********************************************
*/
PROC (0) = sumstats(names,data,fmts1,fmts2,fmtn);
local a, bout;
a   = {"Variable" "Mean" "STD" "MIN" "MAX"};
call outitle(a,fmts1,fmts2);
bout    = names~meanc(data)~stdc(data)~minc(data)~maxc(data);
call outmat(bout,fmts1,fmtn);
endp;
```

Chapter 8

Logit Model

8.1 Data Generation

```
/*
**************************************
*   (C) Copyright 1999, Peter Lenk. All Rights Reserved.
*  GLOGIT2.GSS
*       Generats data for HB LOGIT Regression Model
*---------> Different choice matrices.
*
*       Select one of mvar+1 alternatives where the last alternative is
*       the base brand.
*
*       P(C_{ij} = k|X_{ij}) = exp(X_{ij}[k]'beta_i)/(1 + sum exp(X_{ij}[l]'beta_i))

*       for     i = 1, ..., nsub
*               j = 1, ..., yrows[i]
*               k = 1, ..., mvar+1
*       Alternative mvar is the base vector.
*
*
*       beta_i is rankx vector
*
*       X_{ij} is mvar x rankx
*           X will have brand intercepts for the first mvar-1 brands,
*           and coefficients for price and advertising.
*           To identify the model, we fix the intercept for the last brand to zero.
*
*
*       beta_i  = Theta'Z_i + delta_i
*       delta_i is N(0,Lambda)
*           Z1 is ln(income) and z2 = family size.
*
**************************************
*/
new;
flagplot = 1;                           @ 1 -> do a bunch of plots                      @
nsub    = 100;                          @ Number of subjects                            @
mvar    =  3;                           @ mvar+1 brands                                 @
                                        @ Choice mvar+1 is the base brand               @
yrows   = 20 + ceil(10*rndu(nsub,1));   @ Gives the number of observations per subject  @

ntot    = sumc(yrows);                  @ total number of observations                  @

rankx   = mvar + 2;         @ Brand 1, Brand 2, Brand 3, Price, Advert  @
```

```
rankz   =  3;              @ # rows of Z_i                                                    @
                           @ intercept, ln(income), and family size                           @

@ Define some variable names for Gauss file @
@ Use string arrays                         @
a           = seqa(1,1,mvar);
brands      = 0 $+ "Brand " $+ ftocv(a,1,0);
brands2     = brands|"Base";



xyname      = brands|"Price"|"Advert"|"Choice";  @ | stacks matrices on top of each other @
zname       = {"Constant", "lnIncome",  "HH Size" };
@ To print a character string use: print $ zname; @


@ define two pointers to access xdata matrix @
@ xdata = { x_{11}, ... x_{1n_1}, x_{21}, ..., x_{2,n_2}, ..., x_{nsub,1} ... x_{nsub,n_{nsub}} } @
@ xij = xdata[lxy[i,j]:uxy[i,j],.] @
lxy     = zeros(nsub,maxc(yrows));      @ gives lower subscript @
uxy     = lxy;                          @ gives upper subscript @
s1      = 0;
for i0 (1,nsub,1); i = i0;
    for fj (1,yrows[i],1); j = fj;
        uxy[i,j]    = mvar*(s1+j);
    endfor;
    s1 = s1 + yrows[i];
endfor;
lxy     = uxy - mvar + 1;
lxy = (0-lxy).*(lxy .< 0) + lxy;            @ zero-out the negative entries. @



@ Generate error variance Lambda @


@ Generate error variance Lambda @

@           b1      b2      b3          Price   Advert      @
lambdat = {
            1       .3      -.1         0   0   ,       @ b1        @
            .3      .8      -.05        0   0   ,       @ b2        @
            -.1     -.05    .5          0   0   ,       @ b3        @
```

```
            0       0       0           .2  .1  ,       @ price     @
            0       0       0           .1  .5          @ advert    @
        };
lambdat = 0.01*lambdat;


lbd12   = chol(lambdat);


@ generate Z variables  @
@Z1 is ln(income) @
m       = ln(40000);
s       = (ln(120000) - m)/1.96;
z1      = m + s*rndn(nsub,1);

@Z2 is family size @
z2      = floor(rndnab(nsub,1,3,4,1,10));  @rndnab is my truncated normal(rows, cols, mean, std, a,
zdata   = (z1-meanc(z1))~(z2-meanc(z2));
zdata   = ones(nsub,1)~zdata;




@ Generate theta @
@ beta_i = Theta'z_i + delta)i @

@       B1 - B4     B2-B4   B3-B4   Price   Advertising                 @

thetat  = {
            .5      .3       0      -2      1,      @ Intercept         @
            .8      .3       -.3    .8      -.2,    @ Ln(Income)        @
            -.2     -.2     0       -.3     .5      @ Family Size       @
        };


@ Generate partworths beta @
betat   = zdata*thetat      + rndn(nsub,rankx)*lbd12;



@ generate X & Y data @
ydata   = zeros(ntot*mvar,1);               @ 0/1 choice    @
ydatat  = ydata;                            @ true utilities of Brand j - Brand mvar+1 @
xdata   = zeros(ntot*mvar,rankx);
ipick   = zeros(mvar+1,1);      @ keep track of the number of choices @
```

```
for i0 (1,nsub,1); i = i0;
    for fj (1,yrows[i],1); j = fj;
        @ Do subject i, purchase j @
        @ xij = brands, price, advertising @
        xij = eye(mvar);                        @ Brand Intercepts @
        @ Generate Prices                                          @
        @          Regular Price            Price Promotion        @
        p1      = 5.5 + .1*rndn(1,1) - (rndu(1,1) < .4)*.3;
        p2      = 5.5 + .1*rndn(1,1) - (rndu(1,1) < .4)*.5;
        p3      = 5.2 + .1*rndn(1,1) - (rndu(1,1) < .2)*.2;
        p4      = 5.0 + .05*rndn(1,1);
        price   = (p1-p4)|(p2-p4)|(p3-p4);
        xij     = xij~price;
        @ Do advertising           @
        @ Brand 1 heavily advertises, followed by the other four       @
        a1      = rndu(1,1) < .4;
        a2      = rndu(1,1) < .3;
        a3      = rndu(1,1) < .2;
        a4      = rndu(1,1) < .1;
        advert  = (a1-a4)|(a2-a4)|(a3-a4);
        xij     = xij~advert;

        @ Generate utility for  brands @
        uij = xij*(betat[i,.]');
        uij = uij|0;                @ utility for last brand is 0 @
        pij = exp(uij);
        pij = pij./sumc(pij);
        zij = rndzmn(pij');         @ zij takes values 1 to mvar+1 @
        choice = zeros(mvar,1); @ all zeros indicates last choice @
        if zij <= mvar;
            choice[zij] = 1;
        endif;

        ipick[zij]  = ipick[zij] + 1;


        @ Save it in the data matrix           @
        ydata[lxy[i,j]:uxy[i,j],.]      = choice;
        xdata[lxy[i,j]:uxy[i,j],.]      = xij;

    endfor;
endfor;
xydata          = xdata~ydata;
```

```
create f1 = xpdata with ^xyname, 0, 8;
if writer(f1,xydata) /= rows(xydata);
        errorlog "Conversion of XYDATA to Gauss File did not work";
endif;
closeall f1;

create f1 = zdata with ^zname, 0, 8;
if writer(f1,zdata) /= rows(zdata);
    errorlog "Conversion of ZDATA to Gauss File did not work";
endif;
closeall f1;

save yrows      = yrows;
save lxy        = lxy;
save uxy        = uxy;


save betat      = betat;
save thetat     = thetat;
save lambdat    = lambdat;

@ Define formats for fancy printing @
@ Used to print a matrix of alpha & numeric variables @
let fmt1a[1,3]  = "*.*lf" 10 5;         @ Format for printing numeric variable      @
let fmtsb[1,3]  = "*.*s" 8 8;           @ Format for printing character variable    @
mask    = zeros(mvar+1,1)~ones(mvar+1,1);   @ 0 for alpha, and 1 for numeric            @
fmt1    = fmtsb|fmt1a;                  @ Format for columns of output              @
bout    = brands2~(ipick/ntot*100);
print " Brand    Market Share (%)";
flag        = printfm(bout,mask,fmt1);  @ Formated print.                           @
print;

if flagplot == 1;
_plctrl = -1;           @ use symbols only in plots @
@ plot beta versus ln(income) and family size @
for fj (mvar,rankx,1); j = fj;
    atitle = "Partworth for " $+ xyname[j] $+ " versus " $+ zname[2];
    title(atitle);
    xy(zdata[.,2], betat[.,j]);
    atitle = "Partworth for " $+ xyname[j] $+ " versus " $+ zname[3];
    title(atitle);
    xy(zdata[.,3], betat[.,j]);

endfor;
```

```
graphset;           @ Set plots back to default values @

endif;
end;
```

8.2 Bayesian Analysis

```
/*
************************************
*   (C) Copyright 1999, Peter Lenk. All Rights Reserved.
*   LOGIT2.GSS
*       Select one of mvar+1 alternatives where the last alternative is
*       the base brand.
*---------> Different choice matrices.
*
*       P(C_{ij} = k|X_{ij}) = exp(X_{ij}[k]'beta_i)/(1 + sum exp(X_{ij}[l]'beta_i))

*       for     i = 1, ..., nsub
*               j = 1, ..., yrows[i]
*               k = 1, ..., mvar+1
*       Alternative mvar is the base vector.
*
*
*       beta_i is rankx vector
*
*       X_{ij} is mvar x rankx
*           X will have brand intercepts for the first mvar brands,
*           and coefficients for price and advertising.
*           To identify the model, we fix the intercept for the last brand to zero.
*
*
*       beta_i  = Theta'Z_i + delta_i
*       delta_i is N(0,Lambda)
*
*   Priors:
*       Theta is maxtrix normal(u0,v0).
*       That is, vec(Theta) is N(vec(u0),v0).
*       vec(theta) stacks the columns of theta.
*       Lambda is Inverted Wishart(f0, g0)
**********************************
*/
new;
outfile     = "results1.dat";   @ Specify output file for saving results                  @
                                @ outfile is a string variable that contains a file name   @
inxy        = "xpdata";         @ Name of Gauss file with X, Choice data                   @
inz         = "zdata";          @ Name of Gauss file with Z data                           @
flagtrue    = 1;                @ 1 -> knows true parameters from simulation               @

/*
************************************
```

```
*   Initialize parameters for MCMC
*************************************
*/
smcmc       = 10000;        @ number of iterations to save for analysis                @
skip        = 1;        @ Save every skip iterations                                   @
nblow       = 10000;        @ Initial transition iterations                            @
nmcmc       = nblow + skip*smcmc;   @ total number of iterations                       @
metstd      = 0.3;                  @ STD for random walk metropolis                   @

/*
*************************************
*   Get data
*************************************
*/
@ Get dimensions and pointers @
load yrows  = yrows;    @ Number of observations per subject    @
load lxy    = lxy;      @ xij = xdata[lxy[i,j]:uxy[i,j],.]      @
load uxy    = uxy;

nsub    = rows(yrows);                  @ number of subjects                    @
mvar    = uxy[1,2] - uxy[1,1];          @ Y_{ij} is a mvar vector.              @
ntot    = sumc(yrows);

@ Input Gauss files @
open f1     = ^inxy;                    @ Get Gauss file for X, Y data              @
                                        @ Opens Gauss file & assigns file handle f1 @
xpdata      = readr(f1,rowsf(f1));      @ "p" for picks                             @
            @ readr reads in Gauss file with file handle f1.            @
            @ rowsf(f1) returns the number of rows in the Gauss file.   @
            @ readr reads rowsf(f1) rows, which is the entir dataset.   @
ci          = close(f1);
xpnames     = setvars(inxy);            @ Get the variable names that accompnay X, Y data @
ynames      = xpnames[1:mvar];          @ Use names of interceps for names of components of Y @

rankxp      = cols(xpdata);
rankx       = rankxp - 1;                   @ # of X variables (includes intercept) @
xnames      = xpnames[1:rankx];
@ Last row of xpdata is the choice vector. @

open f1     = ^inz;
zdata       = readr(f1,rowsf(f1));      @ First column of zdata is a vector of ones     @
ci          = close(f1);
znames      = setvars(inz);

rankz   = cols(zdata);                  @ # of Z variables (includes intercept) @
```

```
thdim   = rankx*rankz;                  @ dimension of vec(theta) @

@ Compute some sufficient statistics @
ztz     = zdata'zdata;

/*
*************************************
*   Initialize Priors
*************************************
*/


@ Prior for theta is N(u0,v0) @

u0  = zeros(thdim,1);
v0      = 100*eye(thdim);       @ thdim = rankx*rankz    @
v0i     = invpd(v0);            @ used in updating theta @
v0iu0   = v0i*u0;               @ used in updating theta @



@ Lambda^{-1} is W_rankx(f0,g0 ) @
@ f0 = prior df, g0 = prior scale matrix @
f0      = rankx+2;  f0n = f0 + nsub;
g0i     = eye(rankx);   @ g0^{-1} @

/*
************************************
*   Initialize MCMC
************************************
*/

beta    = zeros(nsub,rankx);
llbeta  = loglike(beta);            @ Compute log-likelihood at beta    @
                                    @ llbeta is used in metropolis      @
theta   = zeros(rankz,rankx);
lambda  = eye(rankx);
lambdai = invpd(lambda);

@ Define data structures for saving iterates & computing posterior means & std @
betam   = zeros(nsub,rankx);    @ posterior mean of beta        @
betas   = zeros(nsub,rankx);    @ posterior std of beta         @
thetag  = zeros(smcmc,thdim);
c       = rankx*(rankx+1)/2;
lambdag = zeros(smcmc,c);       @ save iterations for lambda    @
```

```
/*
************************************
*   Do MCMC
************************************
*/
mixp    = 0;
@ Do the initial transition period @
for i1 (1,nblow,1); imcmc = i1;
    call getlogit;
endfor;


for i1 (1,smcmc,1); imcmc = i1;     @ Save smcmc iterations             @
    for i2 (1,skip,1); jmcmc = i2;      @ Save every skip iterations    @
        call getlogit;
    endfor;
    @ Save the random deviates @
    thetag[imcmc,.] = vecr(theta)';
    betam           = betam     + beta;
    betas           = betas     + beta^2;
    lambdag[imcmc,.]= vech(lambda)';
endfor;


/*
**********************************
*   Compute Posterior Means and STD
**********************************
*/
mixp        = mixp/(nmcmc*nsub);
betam       = betam/smcmc;
thetam      = reshape(meanc(thetag),rankz,rankx);
lambdam     = xpnd(meanc(lambdag));


betas       = sqrt( abs(betas   - smcmc*betam^2)/smcmc);
thetas      = reshape(stdc(thetag),rankz,rankx);
lambdas     = xpnd(stdc(lambdag));
```

```
/*
**********************************
*   Do some output
**********************************
*/
call outputanal;


@ Plot saved iterations against iterations number @
t   = seqa(nblow+skip,skip,smcmc);      @ saved iteration number @
title("Theta versus Iteration");
xy(t,thetag);
title("Lambda versus Iteration");
xy(t,lambdag);
graphset;

end;

/*
**********************************
* GETLOGIT
*   Does one iteration of the HB regression model.
*   INPUT
*       Global Variables
*   OUTPUT
*       Global Variables
**********************************
*/
PROC (0) = getlogit;
local
    thz,bnew,llbnew,i0,i,mui,testp,vibn12,ebin,resid,gni,gn,gn12,w, u;

    thz     = zdata*theta;
    /*
    **********************************
    * Generate beta
    * Use symmetric random walk metropolis
    **********************************
    */
    bnew    = beta + metstd*rndn(nsub,rankx);
    llbnew  = loglike(bnew);                    @ Get log likelihood at new beta @
    u   = ln(rndu(nsub,1));
    @ Do metropolis step  @
    for i0 (1,nsub, 1); i = i0;
```

```
        mui  = thz[i,.]';
        testp = llbnew[i] - llbeta[i]                              @log Likelihood @
            - ( bnew[i,.]' -  mui )'lambdai*( bnew[i,.]' - mui )/2  @ "Priors"      @
            + ( beta[i,.]' -  mui )'lambdai*( beta[i,.]' - mui )/2;

        if u[i] < testp;  @ Accept Candidate @
            mixp        = mixp + 1;
            beta[i,.]   = bnew[i,.];
            llbeta[i]   = llbnew[i];
        endif;          @ else reject candidate & stay pat @
    endfor;

    /*
    *********************************
    * Generate Theta and Lambda from multivariate model:
    *   B = Z*Theta + N(0,Lambda)
    *****************************
    */
    {theta, lambda, lambdai} =
    getmulreg(beta,zdata,ztz,theta,lambda,lambdai,v0i,v0iu0,f0n,g0i);

endp;


/*
*********************************
* GETMULREG
*   Generate multivariate regression parameters.
*       Yd      = Xd*parmat + epsilon
*
*   INPUT
*       yd      = dependent variables
*       xd      = independet variables
*       xdtxd       = xd'xd
*
*       parmat  = current value of coefficient matrix
*       var     = current value of covariance matrix
*       vari        = its inverse
*       v0i     = prior precisions for bmat
*       v0iu0       = prior precision*prior mean for bmat
*       f0n     = posterior df for sigma
*       g0i     = prior scaling matrix inverse for sigma

*
*   OUTPUT
```

```
*       parmat  = updated rankx x mvar coefficient matrix
*       var     = updated variance
*       vari        = updated inverse of sigma
*
*   Calling Statement:
{parmat, var, vari} = getmulreg(yd,xd,xdtxd,parmat,var,vari,v0i,v0iu0,f0n,g0i);
*********************************
*/
PROC (3) = getmulreg(yd,xd,xdtxd,parmat,var,vari,v0i,v0iu0,f0n,g0i);
local vb12, ubn, par, pdim, resid, gni, gn, rp, cp;

    rp      = rows(parmat);
    cp      = cols(parmat);
    pdim    = rp*cp;

    /*
    *********************************
    * Generate parmat from N_{rp x cp}(M,v)
    * par = vecr(parmat)
    * par is N(u,V) whee u = vec(M');
    *   V = (Xd'Xd.*.Var^{-1} + V_0^{-1})^{-1}
    *   u = V*( (Xd'.*.Var^{-1})*vec(Yd') + V_0^{-1}u_0 )
    *******************************
    */

    vb12    = chol(xdtxd.*.vari + v0i);
    ubn     = ( (xd').*.vari )*vecr(yd) + v0iu0;
    par     = cholsol(ubn + vb12'rndn(pdim,1), vb12);
    parmat  = reshape(par,rp,cp);

    /*
    *********************************
    * Generate Var
    *   Var^{-1} is Wishart df = f0n, scale matrix = gn
    *********************************
    */
    resid           = yd - xd*parmat;
    gni             = g0i + resid'resid;
    gn              = invpd(gni);
    {vari, var}     = wishart(cp,f0n,gn);

retp(parmat,var,vari);
endp;
```

```
/*
****************************************
* LOGLIKE
*   Computes multinomial-logit log-likelihood at parameter beta
*   INPUT:
*       beta
*   OUTPUT:
*       Log likelihood
*
****************************************
*/
PROC (1) = loglike(beta);
local  llbeta,i0,i,fj,j,xij,cij,zij,muij,emuij;
llbeta    = zeros(nsub,1);              @ ln(Likelihood x prior) of beta for each subject @
for i0 (1, nsub, 1);    i = i0;                 @ loop over subjects   @
    for fj (1,yrows[i],1);  j = fj;             @ loop over selections @
        xij     = xpdata[lxy[i,j]:uxy[i,j],1:rankx];    @ Get design matrix     @
        cij     = xpdata[lxy[i,j]:uxy[i,j],rankxp];     @ Get vector of choices @
        if maxc(cij) == 0;      @ Picked base brand @
            zij = mvar+1;
        else;                   @ Picked one of brands 1 to mvar @
            zij = maxindc(cij);
        endif;
        muij        = xij*(beta[i,.]');     @ logits   @
        muij        = muij|0;
        emuij       = exp(muij);
        emuij       = sumc(emuij);
        llbeta[i]   = llbeta[i] + muij[zij] -  ln(emuij);
    endfor;
endfor;
retp(llbeta);
endp;


/*
************************************************
*  OUTPUTANAL
*   Does analysis of output and save some results
************************************************
*/
PROC (0) = outputanal;
format 10,5;
local  bout, sout, ebeta, sbeta, cb, rmse, fmts1,fmts2, fmtn1, fmtn2, a,b, flag, i0, i,
betat, thetat, lambdat;
```

```
@ Get true parameters if simulation @
if flagtrue == 1;
    load betat      = betat;
    load thetat     = thetat;
    load lambdat    = lambdat;
endif;


let fmtn1[1,3]  = "*.*lf" 10 5;         @ Format for printing numeric variable      @
let fmtn2[1,3]  = "*.*lf" 10  0;        @ Format for numeric variable, no decimal   @

let fmts1[1,3]  = "-*.*s" 10 9;         @ Format for alpha, left justify            @
let fmts2[1,3]  = "*.*s" 10 9;          @ Format for alpha, right justify           @
format 10,5;                            @ Default pring format                      @


output file = ^outfile reset;   @ outfile is the file handle for the output file    @
                                @ Route printed output to the defined by outfile    @
print "Results from LOGIT1.GSS";
print "Hierarchical Bayes Multivariate LOGIT";
print;
print "Select one of " mvar+1 " alternatives.";
print;
print "beta_i = Theta'z_i + delta_i";
print "delta_i is N(0, Lambda)";
print;
print "Ouput file: " getpath(outfile);      @ File assigned to file handle outfile @
datestr(date);                              @ Print the current data                @
print;
print;
print "-----------------------------------------";
print;
print "MCMC Analysis";
print;
print "Total number of MCMC iterations                 = " nmcmc;
print "Number of iterations used in the analysis       = " smcmc;
print "Number in transition period                     = " nblow;
print "Number of iterations between saved iterations = " skip-1;
print "Proportion of Metroplis Steps Accepted        = " mixp;
print;
print "-----------------------------------------";
print "Number of subjects                 = " nsub;
print "Number of observations per subject:";
print "     Average  = " meanc(yrows);
print "     STD      = " stdc(yrows);
```

```
print "     MIN      = " minc(yrows);
print "     MAX      = " maxc(yrows);
print;
print "Total number of Choices            = " ntot;
print "Number of Alternatives             = " mvar+1;
print "Number of dependent variables X    = " rankx " (including intercept)";
print "Number of dependent variables Z    = " rankz " (including intercept)";
print;



print "Dependent variables are " $xpnames[rankxp];
print "     Summary Statistics for Choices (excluding Base)";
call sumstats(ynames,reshape(xpdata[.,rankxp],ntot,mvar),fmts1,fmts2,fmtn1);
print;
print "First level equation for Logits";
print "logit_{ij} = X_{ij}*beta_i + epsilon_{ij}";
print;
print "     Summary Statistics for X";
call sumstats(xnames[mvar+1:rankx],xpdata[.,mvar+1:rankx],fmts1,fmts2,fmtn1);
print;
print "Second level equation:";
print "beta_i = Theta*z_i + delta_i";
print;
print "     Summary Statistis for Z:";
call sumstats(znames[2:rankz],zdata[.,2:rankz],fmts1,fmts2,fmtn1);
print;



print "-----------------------------------------";
print;
print "Statistics for Individual-Level Regression Coefficients";
if flagtrue == 1;
    ebeta   = meanc(betat);
    sbeta   = stdc(betat);
    print "True Beta";
    sout    = {"Variable" "Mean" "STD"};
    call outitle(sout,fmts1,fmts2);
    bout = xnames~ebeta~sbeta;
    call outmat(bout,fmts1,fmtn1);
endif;

print "HB Estimates of Beta";
sout    = {"Variable" "PostMean" "PostSTD" };
```

```
call outitle(sout,fmts1,fmts2);
ebeta   = meanc(betam);
sbeta   = sqrt( meanc( (betas^2)) + stdc( betam)^2);
bout = xnames~ebeta~sbeta;
call outmat(bout,fmts1,fmtn1);
print;
print "-----------------------------------------";

if flagtrue == 1;
    print "Comparison of True Beta to Individual Level Estimates";
    for i0 (1,rankx,1); i = i0;
        print "Variable " $ xnames[i];
        cb      = corrx( betat[.,i]~betam[.,i] );
        rmse    = betat[.,i] - betam[.,i];
        rmse    = rmse'rmse;
        rmse    = sqrt(rmse/nsub);
        print "Correlation between true and HB  = " cb[1,2];
        print "RMSE between true and HB         = " rmse;
        print;
    endfor;
endif;
print "-----------------------------------------";
print;
print "HB Estimates of Theta";
sout    = "  "~(xnames');
if flagtrue == 1;
    print "True Theta";
    call outitle(sout,fmts1,fmts2);
    bout = znames~thetat;
    call outmat(bout,fmts1,fmtn1);
    print;
endif;
print "Posterior Mean of Theta";
print outitle(sout,fmts1,fmts2);
bout = znames~thetam;
call outmat(bout,fmts1,fmtn1);
print;
print "Posterior STD of Theta";
call outitle(sout,fmts1,fmts2);
bout    = znames~thetas;
call outmat(bout,fmts1,fmtn1);
print;
print "-----------------------------------------";
print;
sout = "  "~(xnames');
```

```
print "HB Estimate of Lambda";
if flagtrue == 1;
    print "True Lambda";
    call outitle(sout,fmts1,fmts2);
    bout = xnames~lambdat;
    call outmat(bout,fmts1,fmtn1);
    print;
endif;
print "Posterior Mean of Lambda";
call outitle(sout,fmts1,fmts2);
bout = xnames~lambdam;
call outmat(bout,fmts1,fmtn1);
print;
print "Posterior STD of Lambda";
call outitle(sout,fmts1,fmts2);
bout = xnames~lambdas;
call outmat(bout,fmts1,fmtn1);
print;
print "=========================================";



output off;
closeall;
endp;




/*
************************************************
* OUTITLE
*   Prints header for columns of numbers.
*   INPUT
*       a   = character row vector of column names
*       fmts1   = format for first column
*       fmts2   = format for second column
*   OUTPUT
*       None
*****************************************
*/
PROC (0) = outitle(a,fmt1,fmt2);
local mask, fmt, flag, ncols;
ncols   = cols(a);
mask    = zeros(1,ncols);
```

```
fmt     = fmt1|(ones(ncols-1,1).*.fmt2);
flag    = printfm(a,mask,fmt);
print;
endp;
/*
************************************************
* OUTMAT
*   Outputs a matrix:
*   (Character Vector)~(Numeric matrix);
*   The entries in the numeric matrix have the same format
*   INPUT
*       bout        = matrix to be printed
*       fmts        = format for string
*       fmtn        = format for numeric matrix
*   OUTPUT
*       None
************************************************
*/
PROC (0) = outmat(bout,fmts,fmtn);
local fmt,mask,flag,ncols, nrows;
ncols       = cols(bout);
nrows       = rows(bout);
fmt         = fmts|(ones(ncols-1,1).*.fmtn);
mask        = zeros(nrows,1)~ones(nrows,ncols-1);
flag        = printfm(bout,mask,fmt);
print;
endp;

/*
************************************************
* SUMSTATS
*   Prints summary statistics for a data matrix
*   INPUT
*       names       = charater vector of names
*       data        = data matrix
*       fmts1       = format for string
*       fmts2       = format for string
*       fmtn        = format for numbers
*   OUTPUT
*       None
************************************************
*/
PROC (0) = sumstats(names,data,fmts1,fmts2,fmtn);
local a, bout;
a   = {"Variable" "Mean" "STD" "MIN" "MAX"};
```

```
call outitle(a,fmts1,fmts2);
bout    = names~meanc(data)~stdc(data)~minc(data)~maxc(data);
call outmat(bout,fmts1,fmtn);
endp;
```