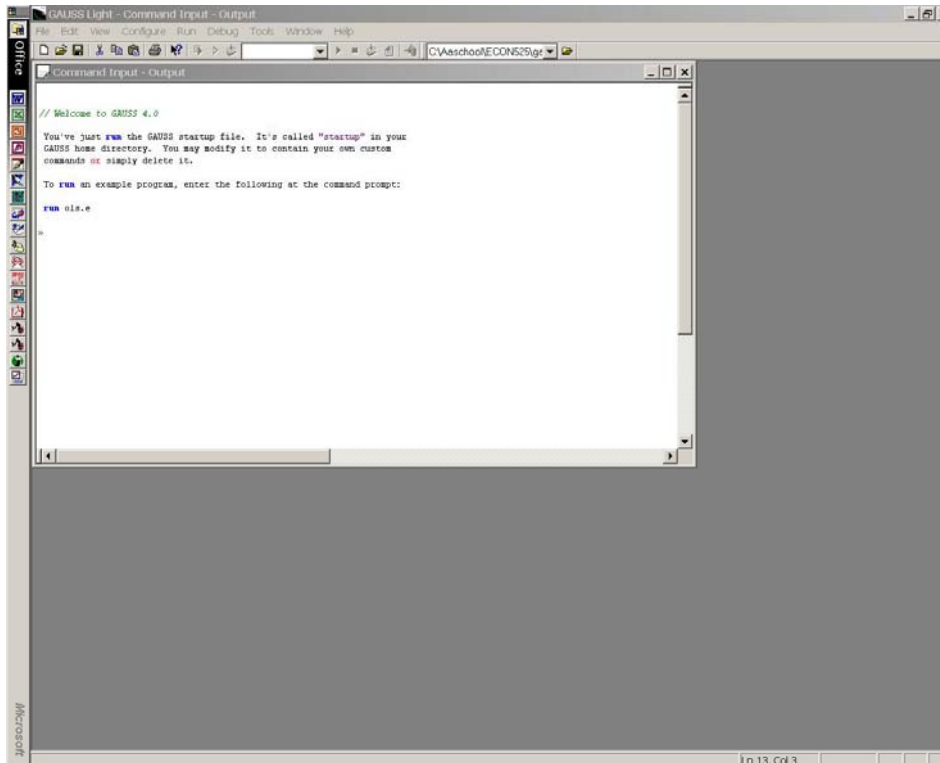


GAUSS Tutorial

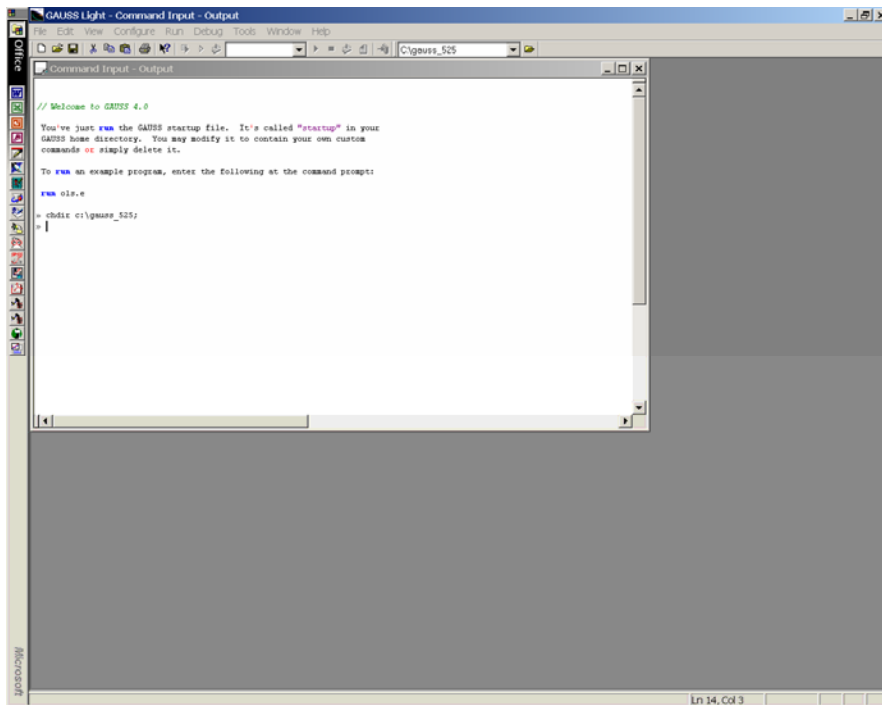
[1] AN EXERCISE ON HOW TO USE GAUSS OR GAUSSLIGHT

- On your computer, create a file folder called “gauss_725”. And download the two files called exer.txt and ols.prg from Dr. Ahn’s website. Locate the two files in the folder, gauss_725. (Or save the two files into a floppy disk). Then, follow the steps described below. The pictures shown below would be slightly different from what you will actually see from your screens.

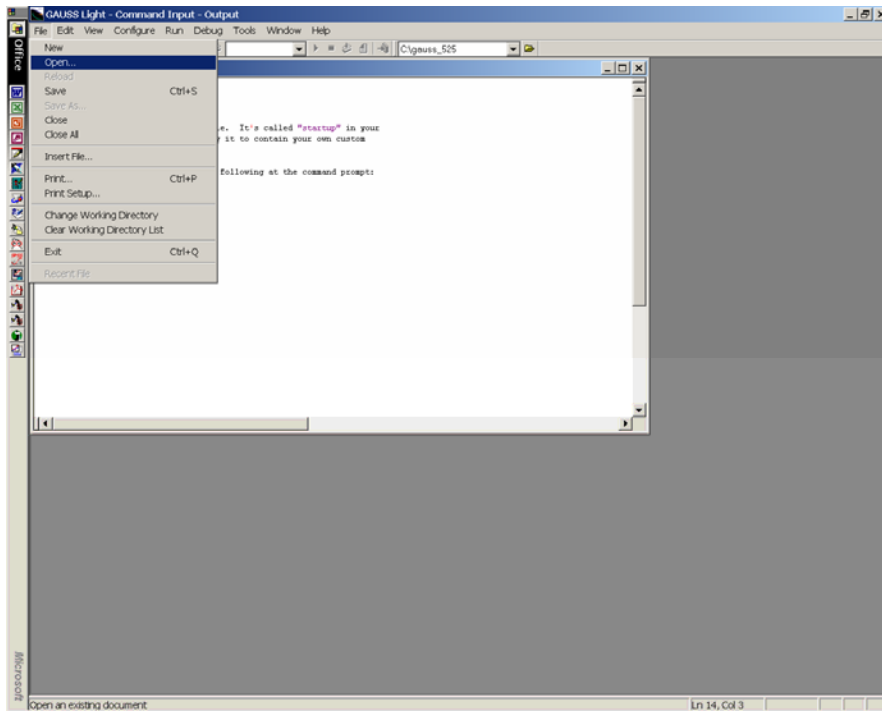
1. Double click on the GAUSS LIGHT 6.0 icon. Then, you will see:



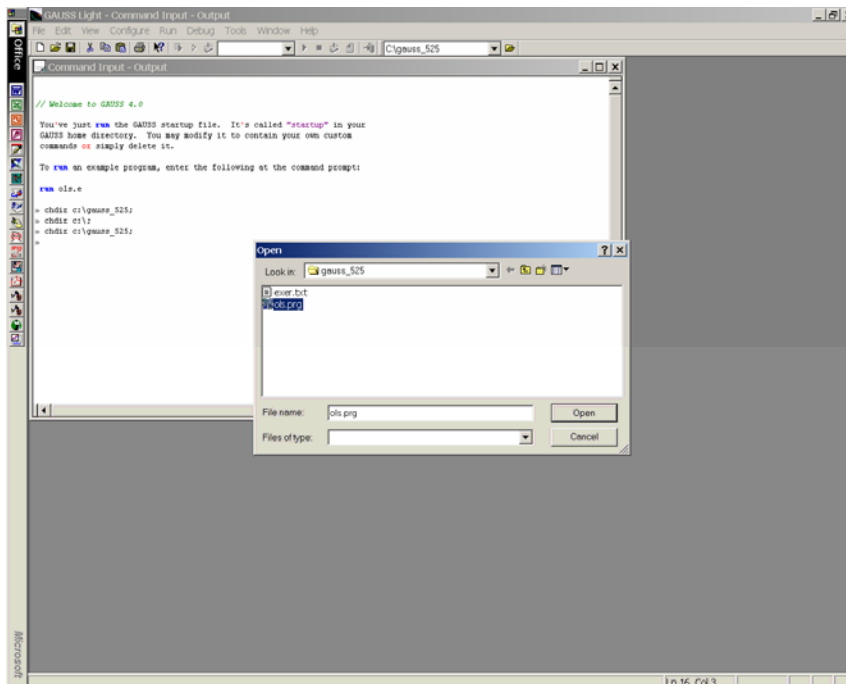
2. Type “chdir c:\gauss_725” (or a: if you use a floppy disk) on the command window and push the return button. By doing so, you can change your working directory to “gauss_725” (or to “a:” if you use a floppy disk).



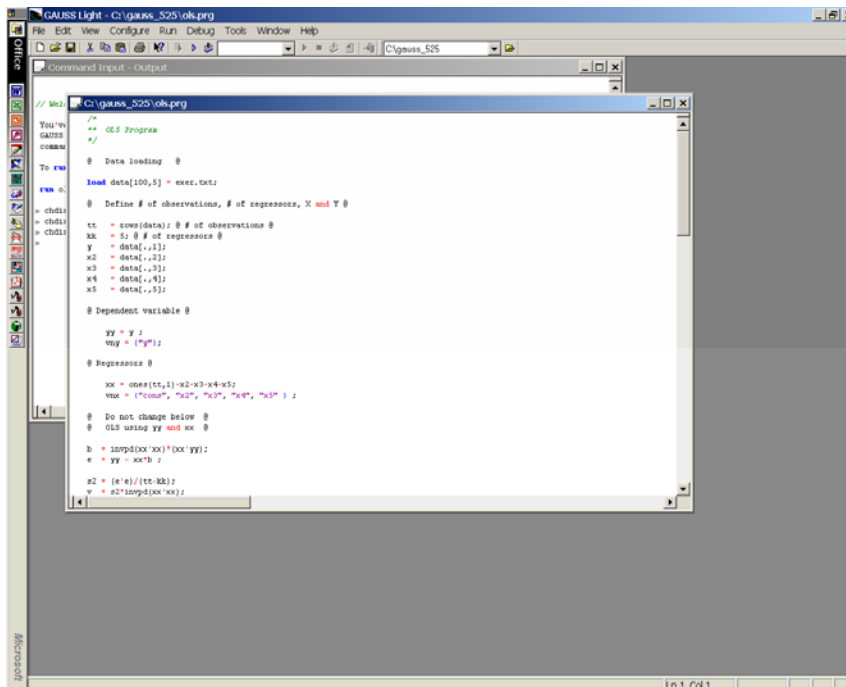
3. Click on **file/open**.



Then, you will see:

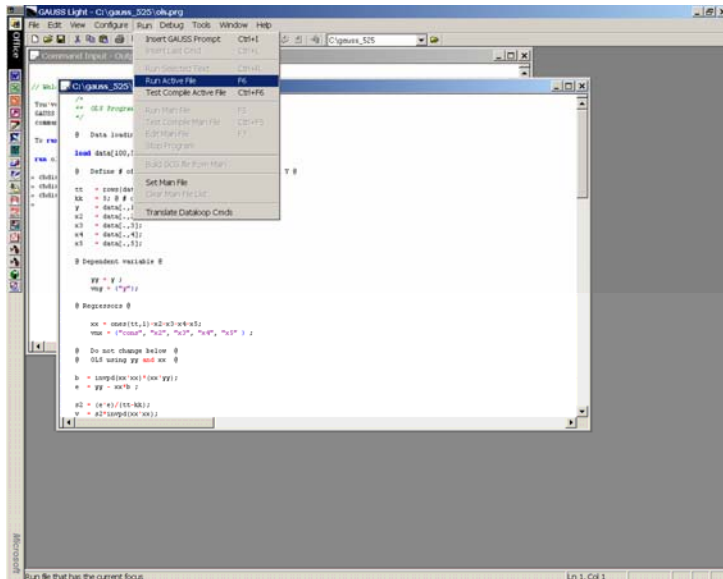


Click on the **Open** button. Then,

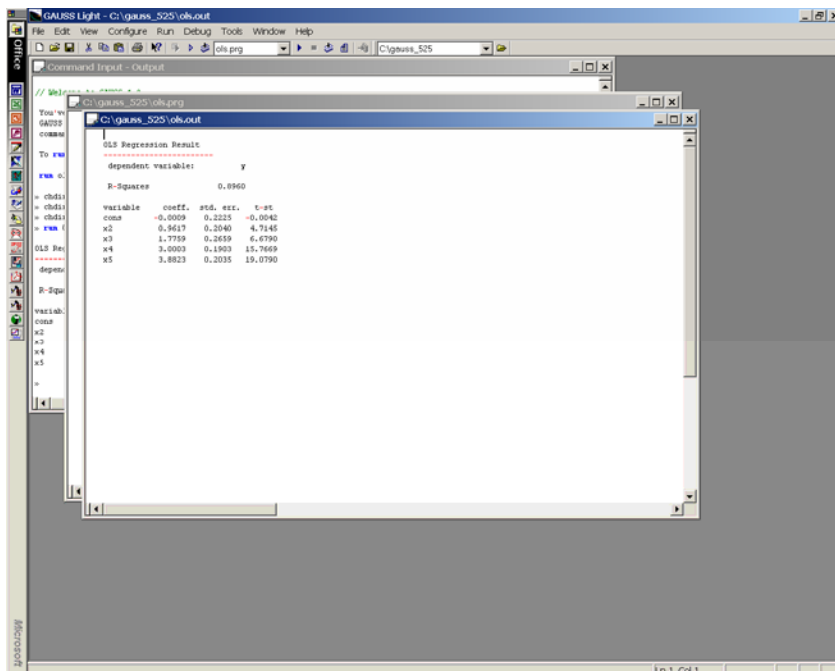


You can edit the program if you want. Once you finish editing, save the program using the save button on the screen. If you would like to create your own program, on **command window**, type, “edit johndoe.prg” (or other name) and push the return key. Then, you can get an empty window named johndoe.prg. Type your own codes on it!

4. To run the program, click on **Action\Run Active File**.



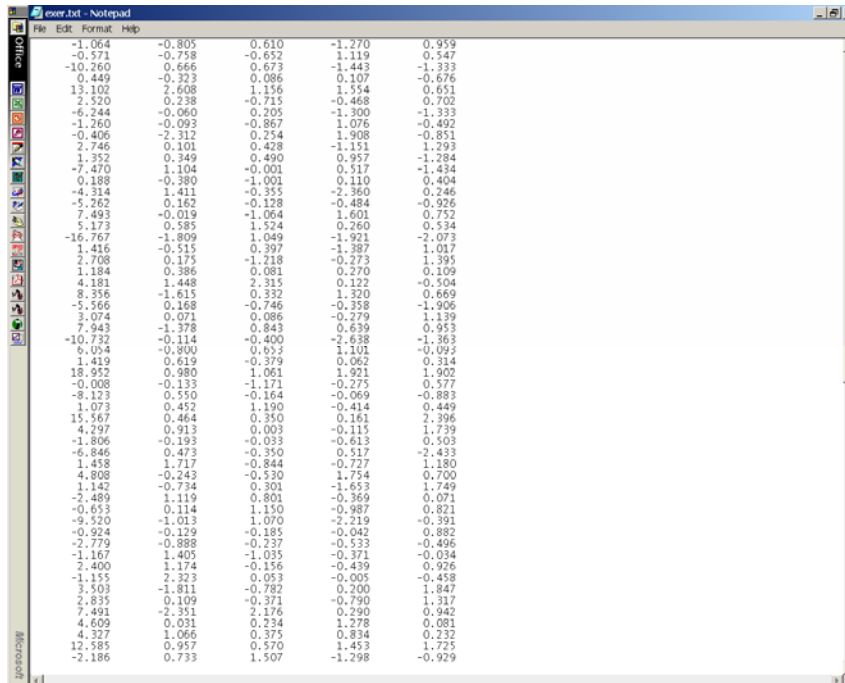
- To see the output file, click on **file/Open**. When the list of the files appears, click on “ols.out”, which is an output file created by running ols.prg. Then, you will see:



[2] SOME USEFUL COMMANDS

(1) Reading data from a text file:

- The form of exer.txt:



The screenshot shows a Notepad window titled 'exer.txt - Notepad'. The window contains a 100x5 matrix of numerical data. The data is organized into five columns and 100 rows. The values range from approximately -2.186 to 18.952. The window's menu bar includes 'File', 'Edit', 'Format', and 'Help'. The Windows taskbar is visible at the bottom, showing the 'Microsoft Office' taskbar and the 'Microsoft' logo.

-1.064	-0.805	0.610	-1.270	0.959
-0.571	-0.758	-0.652	1.119	0.547
-10.260	0.666	0.673	-1.443	-1.333
0.449	-0.323	0.086	0.107	-0.676
13.102	2.608	1.156	1.594	0.651
2.320	0.238	-0.715	-0.468	0.702
-6.244	-0.060	0.205	-1.300	-1.333
-1.260	-0.093	-0.867	1.076	-0.492
-0.406	-2.312	0.254	1.908	-0.851
2.746	0.101	0.428	-1.151	1.293
1.352	0.349	0.490	0.957	-1.284
-7.470	1.104	-0.001	0.517	-1.434
0.188	-0.380	-1.001	0.110	0.404
+4.314	1.411	+0.355	-2.360	0.246
-5.262	0.162	-0.128	-0.484	-0.926
7.493	-0.019	-1.064	1.601	0.752
5.173	0.585	1.524	0.260	0.534
-16.767	-1.809	1.049	-1.921	-2.073
1.416	-0.515	0.397	-1.387	1.017
2.708	0.175	-1.218	-0.273	1.395
1.184	0.386	0.081	0.270	0.109
4.181	1.448	2.315	0.122	-0.504
8.356	-1.615	0.332	1.320	0.669
-5.566	0.168	-0.746	-0.358	-1.906
3.074	0.071	0.086	-0.279	1.139
7.943	-1.378	0.843	0.639	0.953
-10.732	-0.114	-0.400	-2.638	+1.363
6.054	-0.800	0.653	1.101	-0.093
1.419	0.619	-0.379	0.062	0.314
18.952	0.980	1.061	1.921	1.902
-0.008	-0.133	-1.171	-0.275	0.577
-8.123	0.550	-0.164	-0.069	-0.883
1.073	0.452	1.190	-0.414	0.449
15.367	0.464	0.350	0.161	2.396
4.297	0.913	0.003	-0.115	1.739
-1.806	-0.193	-0.033	-0.613	0.503
-6.846	0.473	-0.350	0.517	-2.433
1.458	1.717	-0.844	-0.727	1.180
4.808	-0.243	-0.530	1.754	0.700
1.142	-0.734	0.301	-1.653	1.749
-2.489	1.119	0.801	-0.369	0.071
-0.653	0.114	1.150	-0.987	0.821
-9.520	-1.013	1.070	-2.219	-0.391
-0.924	-0.129	-0.185	-0.042	0.882
-2.779	-0.888	-0.127	-0.533	-0.496
-1.167	1.405	-1.035	-0.371	-0.034
2.400	1.174	-0.156	-0.439	0.926
-1.155	1.323	0.053	-0.005	-0.458
3.503	-1.811	-0.782	0.200	1.847
2.835	0.109	-0.371	-0.790	1.317
7.491	-2.351	2.176	0.290	0.942
4.609	0.031	0.234	1.278	0.081
4.327	1.066	0.375	0.834	0.232
12.585	0.957	0.570	1.453	1.725
-2.186	0.733	1.507	-1.298	-0.929

- This data set contains 100 observations for 5 variables. To read this data set:

```
load data[100,5] = exer.txt;
```

```
y = data[.,1]; @ all the entries on the first column @
```

```
x2 = data[.,2]; @ all the entries on the second column @
```

```
x3 = data[.,3];
```

```
x4 = data[.,4];
```

```
x5 = data[.,5];
```

The last 5 lines define variable names.

(2) Making Comments

GAUSS does not execute any codes between @ and @ or between /* and */. Thus, you can make some comments in the program using @ ... @ or /* ... */.

(3) Matrix Operation

- Let $A = [a_{ij}]$ and $B = [b_{ij}]$ be conformable matrices.
 - $A*B$: Product of A and B.
 - A' : Transpose of A.
 - $A'B$: Product of A' and B.
 - $A[:,1]$: The first column of A.
 - $A[1,:]$: The first row of A.
 - $A[1,2]$: The (1,2)th element of A.
 - $A[1:5,:]$: The 1st, 2nd, 3rd, 4th and 5th rows of A.
 - $A[1 3,:]$: The 1st and 3rd rows of A.
 - $\text{inv}(A)$: Inverse of A.
 - $\text{invpd}(A)$: Inverse of a positive definite matrix A.
 - $\text{diag}(A)$: $n \times 1$ vector of diagonal elements of an $n \times n$ matrix A.
 - $\text{sqrt}(A)$: $[\sqrt{a_{ij}}]$.
 - A^2 : $[a_{ij}^2]$.
 - $A|B$: Merge A and B vertically; $A \sim B$: Merge A and B horizontally.
 - $A.*B$: Kronecker Product of A and B.
 - $\text{sumc}(A)$: $n \times 1$ vector of sums of individual columns for an $m \times n$ matrix A.

$$\text{EX: } A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \rightarrow \text{sumc}(A) = \begin{bmatrix} 9 \\ 12 \end{bmatrix}.$$

- $\text{meanc}(A)$: $n \times 1$ vector of means of individual columns for a $m \times n$ matrix A.

$$\text{EX: } A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \rightarrow \text{meanc}(A) = \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

- $\text{stdc}(A)$: $n \times 1$ vector of standard errors of individual columns for an $m \times n$ matrix A.

- Suppose $A = [a_{ij}]_{m \times n}$; $B = [b_{ij}]_{m \times n}$; $C = [c_{ij}]_{m \times 1}$; and $d = \text{scalar}$.
 - $A./B$: $[a_{ij}/b_{ij}]$ (element by element operation).
 - $A./C$: $[a_{ij}/c_i]$ (element by element operation).
 - $d-A$: $[d-a_{ij}]$.
 - $d*A$: $[da_{ij}]$.
 - A/d : $[a_{ij}/d]$.

- Generating a matrix of standard normally distributed random variables:
 - `rndns(t,k,dd)`: random matrix of `t` rows and `k` columns generated using `dd` as a seed number seed number.
 - For uniformly distributed (between zero and 1) random variables use `rndus[t,k,dd]`.

(4) Do Loop

- When you need to do the same job repeatedly, the following command would be useful:

```
i = 1;
do while i <= 100;
  :::
  i = i + 1;
endo;
```

If you run the above commands, then the codes written on (`:::`) are repeatedly executed.

(5) Drawing Histogram

- Run the following command lines:

```
library pgraph;
graphset;
{a1,a2,a3} = hist(storb, 50);
```

If you execute the three lines, GAUSSLIGHT draw a histogram for the variable named “storb” counting the frequencies for each of 50 categories. The variables named `a1`, `a2` and `a3` will contain some numeric information about the histogram (see the Gauss manual for detail).

- Or, you can run the following command lines:

```
library pgraph;
graphset;
v = seqa(0,0.01,200);
{a1,a2,a3} = hist(storb,v);
```

The v is a 200×1 vector of an additive sequence. The first argument in `sega` is the starting value. The second argument is increment, and the third argument is the number of elements in the sequence. Thus, the vector v is in the form of $(0, 0.1, 0.2, \dots, 1.99)'$. If you execute the above four lines, GAUSSLIGHT draw a histogram for the variable named "storb" counting the frequencies for the entries of v as breakpoints to be used to compute the frequencies.

[3] OLS EXERCISE

Program file: ols.prg

```
/*  
** OLS Program  
*/  
  
@ Data loading @  
  
load data[100,5] = exer.txt;  
  
@ Define # of observations, # of regressors, X and Y @  
  
tt = rows(data); @ # of observations @  
kk = 5; @ # of regressors @  
y = data[:,1];  
x2 = data[:,2];  
x3 = data[:,3];  
x4 = data[:,4];  
x5 = data[:,5];  
vny = {"y"}; @ name of the dependent variable @  
  
@ Dependent variable @  
  
yy = y ;  
vny = {"y"};  
  
@ Regressors @  
  
xx = ones(tt,1)~x2~x3~x4~x5;  
vnx = {"cons", "x2", "x3", "x4", "x5" } ;  
  
@ Do not change below @  
@ OLS using yy and xx @  
  
b = invpd(xx'xx)*(xx'yy);  
e = y - x*b ;  
  
s2 = (e'e)/(tt-kk);  
v = s2*invpd(xx'xx);
```

```
econ = b~sqrt(diag(v))~(b./sqrt(diag(v)));  
econ = vnx~econ;
```

```
se = sqrt(diag(v));
```

```
sst = yy'yy - tt*meanc(yy)^2;  
sse = e'e;
```

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

```
@ Printing out OLS results @
```

```
output file = ols.out reset;
```

```
let mask[1,4] = 0 1 1 1;  
let fmt[4,3] =  
    ".*s" 8 8  
    ".*lf" 10 4  
    ".*lf" 10 4  
    ".*lf" 10 4;
```

```
format /rd 10,4 ;  
"" ;  
"OLS Regression Result" ;  
"-----" ;  
" dependent variable: " $vny ;  
"" ;  
" R-Squares          " r2 ;  
"" ;  
"variable  coeff.  std. err.  t-st " ;  
yyprin = printfm(econ,mask,fmt);  
"" ;
```

```
output off ;
```

Output file: ols.out

OLS Regression Result

dependent variable: y

R-Squares 0.8960

variable	coeff.	std. err.	t-st
cons	-0.0009	0.2225	-0.0042
x2	0.9617	0.2040	4.7145
x3	1.7759	0.2659	6.6790
x4	3.0003	0.1903	15.7669
x5	3.8823	0.2035	19.0790

[4] OLS MONTE CARLO EXPERIMENTS

- Theory says that OLS estimators are normally distributed under ideal conditions. What does it mean?

- Consider the data exer.txt. The data is generated by the following equation:

$$y_t = 0 + x_{t2} + 2x_{t3} + 3x_{t4} + 4x_{t5} + \varepsilon_t, t = 1, \dots, 100, \text{ and } \text{var}(\varepsilon_t) = 4.$$

Observe that $\beta_2 = 1$.

- We now generate a new data set (say, Data Set 1) as follows:

1. For each $t = 1, \dots, 100$, keep the values of x_{t2}, \dots, x_{t5} , but draw ε_t from $N(0,4)$. Then, generate new y_t following the above equation.

2. Estimate $\hat{\beta}_2$. Name this estimate $\hat{\beta}_2^{[1]}$.

3. Generate other data set (say, Data Set 2) by the same way you generate Data Set 1. Then, get $\hat{\beta}_2^{[2]}$.

4. Repeat 5000 (or more) times and get $\hat{\beta}_2^{[1]}, \dots, \hat{\beta}_2^{[5000]}$.

- The econometric theory implies that the estimates will be normally distributed.

- Let check whether it is true or not.

Program file: olsmonte.prg

```
/*  
** Monte Carlo Program  
*/  
  
@ Load Data @  
  
load data[100,5] = exer.txt ;  
  
@ Data generation under Classical Linear Regression Assumptions @  
  
seed = 1;  
tt = 100; @ # of observations @  
kk = 5; @ # of betas @  
iter = 5000; @ # of sets of different data @  
xx = ones(tt,1)~data[.,2:5]; @ Regressors are fixed @  
tb = {0,1,2,3,4} ; @ y = x(2)*1 + x(3)*2 + x(4)*3 + x(5)*4 + e @  
  
storb = zeros(iter,1);  
storse = zeros(iter,1);
```

```

i = 1; do while i <= iter;

@ Generating y @
yy = xx*tb + 2*randns(tt,1,seed);

@ OLS using yy and xx @

b = invpd(xx'xx)*(xx'yy);
e = yy - xx*b ;

s2 = (e'e)/(tt-kk);
v = s2*invpd(xx'xx);

se = sqrt(diag(v));

storb[i,1] = b[2,1];
storse[i,1] = se[2,1];

i = i + 1; endo;

@ Reporting Monte Carlo results @

output file = olsmonte.out reset;

format /rd 12,3;

"Monte Carlo results";
"-----";
"Mean of OLS b(2)           =" meanc(storb);
"s.e. of OLS b(2)          =" stdc(storb);
"mean of estimated s.e. of OLS b(2) =" meanc(storse) ;

library pgraph;
graphset;
{a1,a2,a3}=hist(storb,50);
output off ;

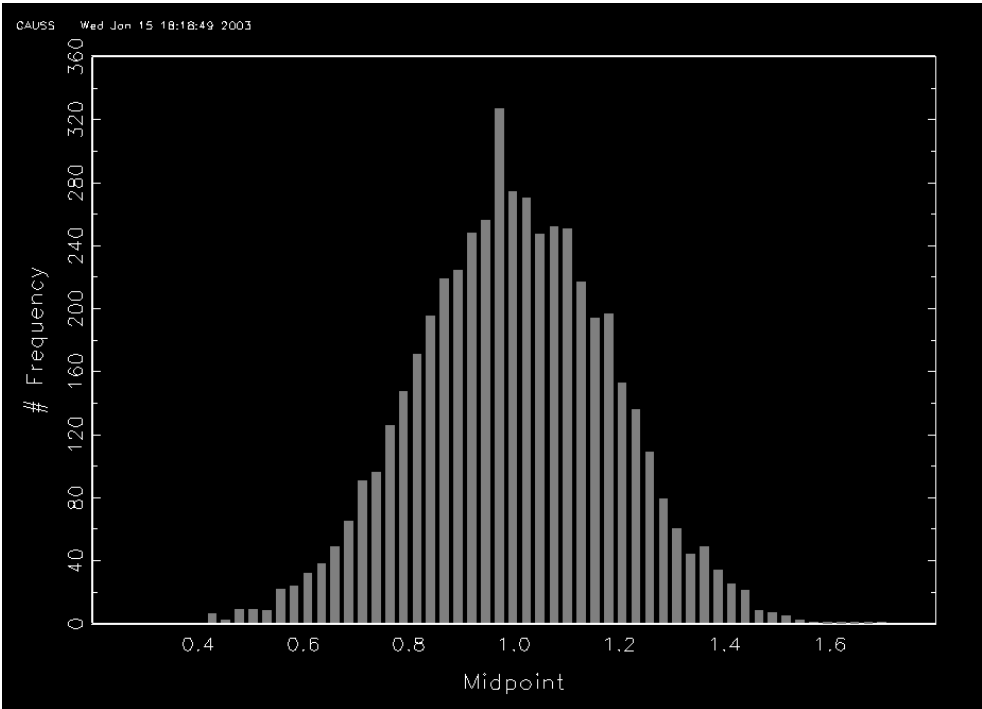
```

Output file: olsmonte.out

Monte Carlo results

Mean of OLS b(2)	=	0.998
s.e. of OLS b(2)	=	0.184
mean of estimated s.e. of OLS b(2)	=	0.185

Graphic Outcome:



[5] OLS MONTE CARLO WHEN REGRESSORS ARE STOCHASTIC

Program file: Wicmonte1.prg

```
/*
** Monte Carlo Program for Weak Ideal conditions I
*/

@ Data generation under Weak Ideal Conditions @

/*
** The regressors are now different across individual data sets.
** That is, regressors are stochastic.
** The errors are different across different data sets
** Errors are normal
*/

@ Model:  $y = x(2)*1 + x(3)*2 + x(4)*3 + x(5)*4 + e$  @

tb = {0,1,2,3,4} ;

seed    = 1;
tt      = 100; @ # of observations @
kk      = 5; @ # of betas @
iter    = 5000; @ # of sets of different data @

storb   = zeros(iter,1);
storse  = zeros(iter,1);
stort   = zeros(iter,1);

i = 1; do while i <= iter;

@ Generating x @

xx = ones(tt,1)~rndns(tt,kk-1,seed) ;

@ Generating y @

yy = xx*tb + 2*rndns(tt,1,seed);

@ OLS using yy and xx @

b = invpd(xx'xx)*(xx'yy);
e = yy - xx*b ;

s2 = (e'e)/(tt-kk);
v = s2*invpd(xx'xx);
```

```

se = sqrt(diag(v));

storb[i,1] = b[2,1];
storse[i,1] = se[2,1];
stort[i,1] = (b[2,1]-1)/se[2,1];

i = i + 1; endo;

@ Reporting Monte Carlo results @

output file = wicmonte1.out reset;

format /rd 12,3;

"Monte Carlo results";
"-----";
"Mean of OLS b(2)                ="   meanc(storb);
"s.e. of OLS b(2)                ="   stdc(storb);
"mean of estimated s.e. of OLS b(2) =" meanc(storse) ;

library pgraph;
graphset;
{a1,a2,a3}=hist(stort,50);
output off ;

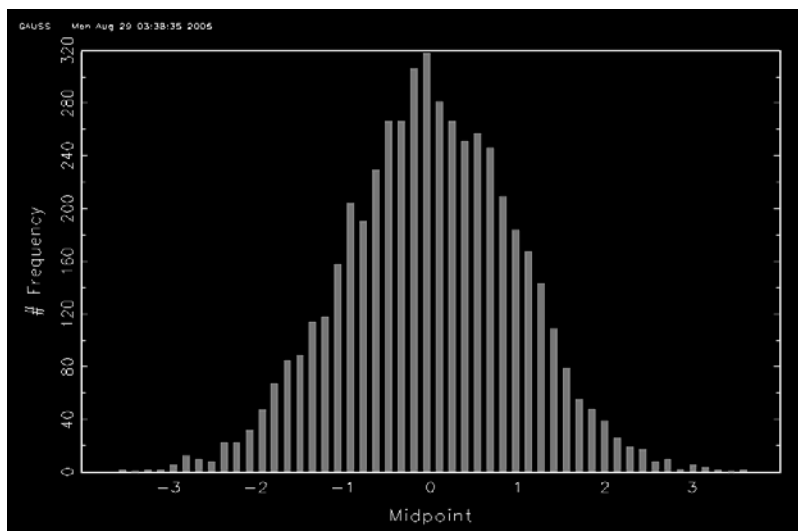
```

Output file: wicmonte.out

```

Monte Carlo results
-----
Mean of OLS b(2)                =          1.000
s.e. of OLS b(2)                =          0.204
mean of estimated s.e. of OLS b(2) =          0.206

```



Program file: wicmonte4.prg

```
/*
** Monte Carlo Program for Weak Ideal conditions IV
*/

@ Data generation under Weak Ideal Conditions @

/*
** The regressors are lagged dependent variables.
** That is, regressors are stochastic.
** The errors are different across different data sets.
** Errors are non-normal: Chi-square(2)-2
*/

@ Model:  $y(t) = \beta(1) + \beta(2)*y(t-1) + e$  @

seed    = 1;
beta2   = 0.5;
beta1   = 0.5;
tt      = 200; @ # of observations @
kk      = 2; @ # of betas @
iter    = 5000; @ # of sets of different data @

storb   = zeros(iter,1);
storse  = zeros(iter,1);
stort   = zeros(iter,1);

i = 1; do while i <= iter;

@ Generating y @

y = 0 ;
j = 1; do while j <= tt;
y = y|(beta1+beta2*y[j,.] + 2*(rndns(1,1,seed)^2+rndns(1,1,seed)^2-
2) );
j = j + 1; endo;

@ OLS using yy and xx @

yy = y[2:tt+1,.]; xx = ones(tt,1)~y[1:tt,.];
b = invpd(xx'xx)*(xx'yy); e = yy - xx*b ;
```

```

s2 = (e'e)/(tt-kk); v = s2*invpd(xx'xx);

se = sqrt(diag(v));

storb[i,1] = b[2,1];
storse[i,1] = se[2,1];
stort[i,1] = (b[2,1]-beta2)/se[2,1];

i = i + 1; endo;

@ Reporting Monte Carlo results @

output file = wicmonte4.out reset;
format /rd 12,3;

"Monte Carlo results";
"-----";
"Mean of OLS b(2) = meanc(storb);
"s.e. of OLS b(2) = stdc(storb);
"mean of estimated s.e. of OLS b(2) = meanc(storse) ;

library pgraph;
graphset;
{a1,a2,a3}=hist(stort,50);
output off ;

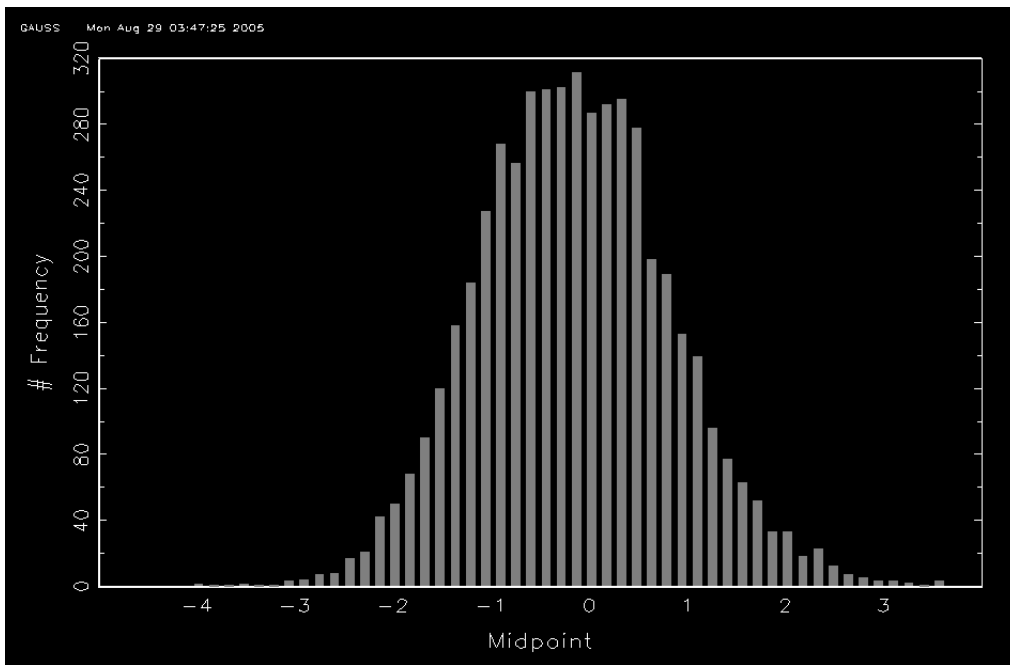
```

Output file: wicmonte4.out

```

Monte Carlo results
-----
Mean of OLS b(2) = 0.488
s.e. of OLS b(2) = 0.061
mean of estimated s.e. of OLS b(2) = 0.062

```



[6] 2SLS MONTE CARLO

Program file: tslsmonte.prg

```
/*
** Monte Carlo Program for 2SLS and Hausman Tests
*/

@ Data generation under Strong Ideal Conditions @

/*
** True Model:  $y = \beta(1) + \beta(2)*x^* + e$ 
** Observed  $x = x^* + v$ 
** Instruments:  $z1, z2$ 
**  $x^* = 1 + \rho*z1 + \rho*z2 + u, u \text{ iid } N(0,1)$ 
*/

/*
** By running this program, we will learn:
** (i) effects of the size of measurement errors on OLS
** (ii) consistency of 2SLS estimator even under measurement errors
** (iii) the weak instruments problem.
*/

seed = 1;
beta2 = 0.5;
beta1 = 0.5;
vvar = 4; @ variance of measurement errors @
rho = 1; @ correlation between x and z @

tt = 100; @ # of observations @
kk = 2; @ # of betas @
iter = 3000; @ # of sets of different data @

storb = zeros(iter,1);
storse = zeros(iter,1);
stort = zeros(iter,1);

stortb = zeros(iter,1);
stortse = zeros(iter,1);
stortt = zeros(iter,1);

hausman1 = zeros(iter,1);
hausman2 = zeros(iter,1);
```

```

i = 1; do while i <= iter;

@ Generating y and x with measurement errors @

z1 = rndns(tt,1,seed) ;
z2 = rndns(tt,1,seed) ;
xt = 1 + rho*z1 + rho*z2 + rndns(tt,1,seed);
mea = rndns(tt,1,seed);
y = beta1 + beta2*xt + 2*rndns(tt,1,seed);
x = xt + sqrt(vvar)*mea ;

@ OLS using yy and xx @

yy = y;
xx = ones(tt,1)~x;

b = invpd(xx'xx)*(xx'yy);
e = yy - xx*b ;

s2 = (e'e)/(tt-kk);
v = s2*invpd(xx'xx);

se = sqrt(diag(v));

storb[i,1] = b[2,1];
storse[i,1] = se[2,1];
stort[i,1] = (b[2,1]-beta2)/se[2,1];

@ 2SLS using zz as instruments @

zz = ones(tt,1)~z1~z2;

xpx = (xx'zz)*invpd(zz'zz)*(zz'xx) ;
xpy = (xx'zz)*invpd(zz'zz)*(zz'yy) ;
tb = invpd(xpx)*xpy;
te = yy - xx*tb ;

ts2 = (te'te)/(tt-kk);
tv = ts2*invpd(xpx);

tse = sqrt(diag(tv));

stortb[i,1] = tb[2,1];
stortse[i,1] = tse[2,1];
stortt[i,1] = (tb[2,1]-beta2)/tse[2,1];

```

```

@ Hausman Test: Original @

haus = (tb-b)'pinv(s2*invpd(xpx)-s2*invpd(xx'xx))*(tb-b) ;
df    = rank(invdpd(xpx)-invpd(xx'xx)) ;
pval  = cdfchic(haus,df);
if pval >= 0.05; hausman1[i,1]= 0; else; hausman1[i,1] = 1; endif;

@ Hausman Test: t-test version @

res   = x - zz*invpd(zz'zz)*(zz'x);
xres  = xx~res ;
augb  = invpd(xres'xres)*(xres'yy);
auge  = yy - xres*augb ;
aug2  = (auge'auge)/(tt-cols(xres)) ;
augv  = aug2*invpd(xres'xres);

haus  = augb[3,1]/sqrt(augv[3,3]);
df    = tt-4;
pval  = 2*cdftc(abs(haus),df);
if pval >= 0.05; hausman2[i,1]= 0; else; hausman2[i,1] = 1; endif;

i = i + 1; endo;

@ Reporting Monte Carlo results @

output file = tslsmonte.out reset;

format /rd 12,3;

"Monte Carlo results";
"-----";
"Mean of OLS b(2)           ="  meanc(storb);
"s.e. of OLS b(2)         ="  stdc(storb);
"mean of estimated s.e. of OLS b(2) ="  meanc(storse) ;
"-----";
"Mean of 2SLS b(2)        ="  meanc(stortb);
"s.e. of 2SLS b(2)       ="  stdc(stortb);
"mean of estimated s.e. of 2SLS b(2) ="  meanc(stortse);
"-----";
"Rejection Rate of Hausman Test 1  ="  meanc(hausman1);
"Rejection Rate of Hausman Test 2  ="  meanc(hausman2);

output off ;

```

Output file: tslsmonte.out

Monte Carlo results

Mean of OLS b(2)	=	0.214
s.e. of OLS b(2)	=	0.082
mean of estimated s.e. of OLS b(2)	=	0.080

Mean of 2SLS b(2)	=	0.503
s.e. of 2SLS b(2)	=	0.168
mean of estimated s.e. of 2SLS b(2)	=	0.166

Rejection Rate of Hausman Test 1	=	0.606
Rejection Rate of Hausman Test 2	=	0.607

[7] NLLS EXERCISE

Program file: nlls.prg

```
/*
** Probit MLE Estimation
*/

new ;

@ LOADING DATA @

    load dat[935,17] = wage2.txt ;

@ OPEN OUTPUT FILE @

    output file = nlls.out reset ;

@ DEFINE VARIABLES @

    wage      = dat[.,1] ;
    hours     = dat[.,2] ;
    iq        = dat[.,3] ;
    kww       = dat[.,4] ;
    educ      = dat[.,5] ;
    exper     = dat[.,6] ;
    tenure    = dat[.,7] ;
    age       = dat[.,8] ;
    married   = dat[.,9] ;
    black     = dat[.,10] ;
    south     = dat[.,11] ;
    urban     = dat[.,12] ;
    sibs      = dat[.,13] ;
    brthord   = dat[.,14] ;
    meduc     = dat[.,15] ;
    feduc     = dat[.,16] ;
    lwage     = dat[.,17] ;

@ DEFINE # of OBSERVATIONS @

    tt = rows(dat) ;
```



```

@ DEFINE MODEL ERROR TERM @

    yy = wage;
    vny = {"wage"};

proc res(b) ;
local e      ;

    e = yy - exp(b[1] + b[2]*educ + b[3]*exper) ;

retp( e ) ;
endp ;

@ DEFINE INITIAL VALUE @

    bb = {0,0,0} ;

library optmum;
#include optmum.ext;
optset ;

proc f(b) ;
local ltt ;

    ltt = res(b)'res(b) ;

retp( ltt ) ;
ENDP ;

b0 = bb ;
__title = "NLLS";
__opgtol = 1e-4;
__opstmth = "bfgs, half";
__output = 0 ;

{b,func,grad,retcode} = optmum(&f,b0) ;

@ Covariance matrix @

    hh = - gradp(&res,b);
    s2 = func/tt;
    cov = s2*invpd(hh'hh);
    se = sqrt(diag(cov));
    r2 = 1- func/(yy'yy-tt*meanc(yy)^2);

```

```

format /rd 10,4 ;
" " ;
"NLLS Estimation Result" ;
"-----" ;
" dependent variable: " $vny ;
" " ;
" R-square:      " r2 ;
" " ;
"   coeff.      std. err.   t-st " ;
b~se~(b./se);
" " ;

output off ;

```

Output file: nlls.out

```

NLLS Estimation Result
-----
dependent variable:      wage

R-square:      0.1372

   coeff.      std. err.   t-st
5.5764      0.1130      49.3474
0.0777      0.0062      12.4974
0.0199      0.0034      5.8440

```