```
new ;
closeall ;

//  ************************************************************************
//
//  incae_tutorial_02_estimation_singleagent.gss
//
//  Program for the Solution, Simulation, and Estimation of a single-agent
//  dynamic model of market entry-exit
//
//  Given parameters fixed by user, the program:
//      (a) computes the solution of the dynamic programming (DP) problem
//      (b) generates a simulated dataset from the solution of the model
//      (c) uses simulated (as if it were actual data) to estimate parameters of
the model
//
//  The estimation method is:
//      (a) Two-step Least Squares using finite dependence property
//
//  San Jose, Costa Rica, June 2022
//
//  ************************************************************************

library pgraph ;

// --------------
//  OUTPUT FILE
// --------------
wdir = "C:\\Users\\victo\\Dropbox\\documents\\COURSES\\2022_INCAE\\TUTORIALS\\" ;
// Working directory
buff = changedir(wdir) ;
fileout  = "incae_tutorial_02_estimation_singleagent.out" ; // Name of output file
output file = ^fileout reset ;
format /mb1 /ros 16,4 ;

// ----------------------------------------------------------------
//  Model:
//
//      y = 0 ----> No active in market
//      y = 1 ----> Active in market
//
//          x = (y[t-1], s[t]), where s[t] represents market size
//
//  Parameters:
//      ecost  =  Entry cost
//      fcost =   Fixex cost
//      bsize =   Slope parameter for market size
//
// ----------------------------------------------------------------
//  Profit function:
```

```
//
//      Profit_i(y=0) = 0
//
//      Profit_i(y=1) = bsize * s[t] - fcost - ecost * (1-y[t-1]) - eps[t]
//
//  where eps[t] is i.i.d. over time with Logistic distribution
//
//  Discount factor = delta
//
//  fs(s[t+1] | s[t]) = Transition probability of market size
//  This transition probability comes from a discretized version of an AR(1)
process:
//      s[t+1] = as_0 + as_1 * s[t] + as_2 * e[t], where e[t] is i.i.d. N(0,1)
//
// ---------------------------------------------------------------

// ------------------------------------------------
//  1. Values of parameters and other constants
// ------------------------------------------------

nobs    = 1000 ;    // Number of individuals in simulated dataset
                    // The "panel dataset" has 1 observation per individual (T=1)
                    // For each individual i, the dataseyt includes y[i,0],
y[i,1], s[i,1]
                    // It is simple to extend the code to consider a "true" panel
with T>1.

ecost = 1.0 ;    // Entry cost
fcost = 0.5 ;    // Fixed cost
bsize = 1.0 ;    // Slope parameter for market size

as_0 = 0.10 ;   // AR(1) process market size: intercept
as_1 = 0.90 ;    // AR(1) process market size: slope
as_2 = 0.10 ;     // AR(1) process market size: std. dev. of shock

numsize = 101; // Number of cells in discretization of market size
minsize = 0;  // Minimum value insupport of market size
maxsize = 2;  // Minimum value insupport of market size
step = (maxsize - minsize)/(numsize - 1); // step size in the uniform-grid
discretization
valsize = seqa(minsize, step, numsize); // Grid of values in discretization of
market size

delta = 0.95 ;    // Discount factor

trueparam = (ecost | fcost | bsize) ;  // Vector values of parameters in profit
function
namesb = "Entry Cost"
       | "Fixed Cost"
       | "Slope Msize"  ; // Vector with names of parameters
```

```
kparam  = rows(trueparam) ;   //  Number of parameters to estimate

// Trasition matrix for Market size
lowthres = (-1e36 | (valsize[1:numsize-1]+valsize[2:numsize])/2 );
higthres = ( (valsize[1:numsize-1]+valsize[2:numsize])/2 | 1e36) ;

ftran = cdfn((higthres' - as_0 - as_1 * valsize)./as_2) - cdfn((lowthres' - as_0 -
as_1 * valsize)./as_2) ; // Transition with discretization
sumcols_ftran = sumc(ftran');
ftran = ftran./(sumcols_ftran') ;  // Normalization such that every row sums
exactly

plotXY(valsize,ftran[51:55,.]');

// ------------------------------------------------
//  2. Solving for Integrated Value Function
//      Using Fixed point iterations in
//      Integrated Bellman equation
// ------------------------------------------------

profit_0 =  bsize * valsize - fcost - ecost ; // Vector with profits of an active
firm when y[t-1]=0
profit_1 =  bsize * valsize - fcost ; // Vector with profits of an active firm
when y[t-1]=1

cconv = 1e-12 ; // Concergence constant
criter = 1000 ; // Initial value for convergence criterion. It should be greater
than cconv
value_0 = zeros(numsize,1) ; // Initialization of value function for y[t-1]=0
value_1 = zeros(numsize,1) ; // Initialization of value function for y[t-1]=1

iter=1 ;
do while (criter>cconv) ;
  "Value function iteration  =";; iter ;
  "    Convergence criterion =";; criter ;
  "" ;
  ev_0 = ftran * value_0 ;
  ev_1 = ftran * value_1 ;

  // Updating value function
  newvalue_0  = delta * ev_0 + ln(1 + exp(profit_0 + delta*(ev_1-ev_0))) ;
  newvalue_1  = delta * ev_0 + ln(1 + exp(profit_1 + delta*(ev_1-ev_0))) ;

  criter = maxc(abs(newvalue_0-value_0) | abs(newvalue_1-value_1)) ; // Sup-norm
for convergence criterion
  value_0 = newvalue_0 ;
  value_1 = newvalue_1 ;
  iter=iter+1 ;
endo ;
```

```
// Plotting Value function

plotXY(valsize,value_0~value_1);

// ------------------------------------------------
//  3. Calculating CCP Function for entry
// ------------------------------------------------
ccp_0 = profit_0 + delta * ftran * (value_1 - value_0);
ccp_0 = 1./(1+exp(-ccp_0)) ;

ccp_1 = profit_1 + delta * ftran * (value_1 - value_0);
ccp_1 = 1./(1+exp(-ccp_1)) ;

myopic_ccp_0 = 1./(1+exp(-profit_0)) ;
myopic_ccp_1 = 1./(1+exp(-profit_1)) ;

// ------------------------------------------------
//  4. Figures with CCPs
// ------------------------------------------------

//Declare plotControl structure
struct plotControl myPlot;

//Initialize plotControl structure
myPlot = plotGetDefaults("scatter");

//Set labels, location, and orientation of legend
label = "Forward-Looking P(1 | y[t-1]=0)" $| "Myopic P(1 | y[t-1]=0)";
location = "top left";
orientation = 0;

plotSetLegend(&myPlot, label, location, orientation);

plotXY(myplot, valsize, ccp_0~myopic_ccp_0);

//Set labels, location, and orientation of legend
label = "Forward-Looking P(1 | y[t-1]=0)" $| "Forward-Looking P(1 | y[t-1]=1)";
location = "top left";
orientation = 0;
plotSetLegend(&myPlot, label, location, orientation);

plotXY(myplot, valsize, ccp_0~ccp_1);


//
-------------------------------------------------------------------------------
// 5. COMPUTING THE STEADY-STATE DISTRIBUTION OF THE STATE VARIABLES (y[t-1],
s[t])
//
//      I will use this distribution to draw the initial value (y[i,0], s[i,1])
```

```
//      of the vector of state variables in the simulated sample.
//      This is not strictly necessary, and we could initialize the simulate data
//      with arbitrary values for (y[i,0], s[i,1]), but using the ergodic or
//      steady-state distribution is more consistent with the model.
//      *** More about this in class
//
--------------------------------------------------------------------------------
//
//      - In a model without market size s[t], and psteady = Steady state
Prob(y[t-1]=1):
//          psteady = psteady * CCP(1|1) + (1-psteady) * CCP(1|0)
//      Solving of psteady:
//          psteady = CCP(1|0) / (1 - CCP(1|1) + CCP(1|0))
//
//      - We can proceed similarly in a model with market size s[t].
//          psteady(1,s') = sum_over_s{ psteady(0,s) * CCP(1|0,s) * ftran(s' | s)
}
//                        + sum_over_s{ psteady(1,s) * CCP(1|1,s) * ftran(s' | s)
}
//      - And similarly:
//          psteady(0,s') = sum_over_s{ psteady(0,s) * CCP(0|0,s) * ftran(s' | s)
}
//                        + sum_over_s{ psteady(1,s) * CCP(0|1,s) * ftran(s' | s)
}
//
//      - In compact matrix form, the vector psteady with all probabilities
psteady(y,s):
//                  psteady = Ftran_ys' * psteady
//      where Ftran_ys is the joint transition probability matrix Pr( y[t], s[t+1]
| y[t-1], s[t] )
//
--------------------------------------------------------------------------------

// 5.1. We start constructing the joint transition probability Ftran_ys
//      Pr( y[t], s[t+1] | y[t-1], s[t] ) = Pr( y[t] | y[t-1], s[t]) * Pr ( s[t+1]
| s[t] )
//                        = CCP( y[t] | y[t-1], s[t]) * ftran ( s[t+1] | s[t] )

ftran_ys_00 = (1-ccp_0') .* ftran ;
ftran_ys_01 = (ccp_0') .* ftran ;
ftran_ys_10 = (1-ccp_1') .* ftran ;
ftran_ys_11 = (ccp_1') .* ftran ;

ftran_ys = ( ftran_ys_00 ~ ftran_ys_01 )
         | ( ftran_ys_10 ~ ftran_ys_11 );


 // 5.2. We now solve the system of equations:
 //              psteady = Ftran_ys' * psteady
 //      This is a fixed-point mapping, it is a contraction, and we can
```

```
 //      obtain its unique solution by using fixed-point iterations

cconv = 1e-6 ;
criter = 1000 ;
psteady = (1/(2*numsize))*ones(2*numsize,1) ;
do while criter>cconv ;
  "Criter =";; criter ;
  pbuff = ftran_ys' * psteady ;
  criter = maxc(abs(pbuff-psteady)) ;
  psteady = pbuff ;
endo ;

// For convenience, we can split vector psteady in two vectors:
//      psteady_0 with probs for y[t-1]=0
//      psteady_1 with probs for y[t-1]=1
psteady_0 = psteady[1:numsize];
psteady_1 = psteady[numsize+1:2*numsize];

// Plot of steady-state distribution of the state variables
plotXY(valsize, psteady_0 ~ psteady_1);


// ------------------------------------------------------------------------
//  6. PROCEDURE TO SIMULATE DATA FROM THE SOLUTION
// ------------------------------------------------------------------------

// 6.1. Getting random draws
//      from the steady-state distribution
//      of the state variables (y[t-1], s[t])

// Thresholds in the unit interval that determine the interval
// for each value of the state
pbuff1 = cumsumc(psteady) ;
pbuff0 = cumsumc((0|psteady[1:2*numsize-1])) ;

// Vector with Uniform[0,1] random draws
uobs = rndu(nobs,1) ;

// Vector with simulated states (y[t-1], s[t]) at t=1
// Each state (y[t-1, s[t]) is represented with an index
// between 1 and 2*numsize.
// That is: state = 1 corresponds to (y[t-1], s[t]) = (0, s_1)
//      ... state = numsize corresponds to (y[t-1], s[t]) = (0, s_numsize)
//          state = numsize + 1 corresponds to (y[t-1], s[t]) = (1, s_1)
//      ... state = 2 * numsize corresponds to (y[t-1], s[t]) = (1, s_numsize)

sim_ys = ((uobs.>=(pbuff0')).*(uobs.<=(pbuff1'))) * seqa(1,1,2*numsize) ;

sim_y0 = (sim_ys .>= numsize);
sim_s1 = (sim_y0.==0) .* sim_ys + (sim_y0.==1) .* (sim_ys - numsize);
```

```
// 6.2. Getting random draws from individuals' choices at t=1
//      from the CCPs conditional on state (y[0],s[1])

// New random draws from Uniform[0,1]
// These random draws are for simulation y[1]

uobs = rndu(nobs,1) ;

// Generating simulations for y[1] conditional on (y[0],s[1])

sim_y1 = (sim_y0.==0) .*  (uobs .<= ccp_0[sim_s1])
       + (sim_y0.==1) .*  (uobs .<= ccp_1[sim_s1]) ;


// ------------------------------------------------------
//  7.  NONPARAMETRIC FREQUENCY ESTIMATOR OF CCP FUNCTION
// ------------------------------------------------------

// Mean value of y1 when y0=0 and when y0=1
meany1_0 = sumc(sim_y1.*(sim_y0.==0))/sumc(sim_y0.==0) ;
meany1_1 = sumc(sim_y1.*(sim_y0.==1))/sumc(sim_y0.==1) ;


// 7.1. Initializing vectors that will contain the estimates of CCPs
npest_ccp_0 = zeros(numsize,1) ;
npest_ccp_1 = zeros(numsize,1) ;

// 7.2. Loop for calculating frequency estimates of all CCPs
s=1 ;
do while s<=numsize ;
  denom_0 = sumc( (sim_y0.==0) .* (sim_s1.==s) ) ;
  numer_0 = sumc( sim_y1 .* (sim_y0.==0) .* (sim_s1.==s) ) ;
  if (denom_0==0) ;
    npest_ccp_0[s] = meany1_0 ;
  else ;
    npest_ccp_0[s] = numer_0/denom_0 ;
  endif ;

  denom_1 = sumc( (sim_y0.==1) .* (sim_s1.==s) ) ;
  numer_1 = sumc( sim_y1 .* (sim_y0.==1) .* (sim_s1.==s) ) ;
  if (denom_1==0) ;
    npest_ccp_1[s] = meany1_1 ;
  else ;
    npest_ccp_1[s] = numer_1/denom_1 ;
  endif ;
  s=s+1 ;
endo ;

// 7.3. Plot of true and estimated CCPs
```

```
plotXY(valsize, ccp_0 ~ npest_ccp_0);

plotXY(valsize, ccp_1 ~ npest_ccp_1);

// ------------------------------------------------------------------------
//  8. TWO-STEP LEAST SQUARES ESTIMATION
//     USING FINITE DEPENDENCE PROPERTY OF
//     MARKET ENTRY-EXIT MODEL
// ------------------------------------------------------------------------
//
// The regression-like model has the following form:
//
//     W[t] = theta_1 + theta_2 * s[t] + theta_3 * (1-y[t-1]) + e[t]
//
// with: theta_1 = - fcost; theta_2 = bsize; theta_3 = - ecost
// and:
//  W[t] = ln( CCP(1|y[t-1],s[t]) ) -  ln( CCP(0|y[t-1],s[t]) )
//       + delta * sum_over_all_s[t+1]
//      {  [ln( CCP(0|1,s[t+1]) ) - ln( CCP(0|0,s[t+1]) )] * ftran(s[t+1]|s[t]) }
// ------------------------------------------------------------------------

// 8.1. Constructing vector of observations for variable W[t]
//      We distinguish two components: W[t] = W1[t] + W2[t], with:
//
//       W1[t] = ln( CCP(1|y[t-1],s[t]) ) -  ln( CCP(0|y[t-1],s[t]) )
//
//       W2[t] = delta * sum_over_all_s[t+1]
//      {  [ln( CCP(0|1,s[t+1]) ) - ln( CCP(0|0,s[t+1]) )] * ftran(s[t+1]|s[t]) }


// - Since we use frequency estimates of CCPs, some of the estimates
// are exactly equal to 0 or to 1.
// - This generates problems when calculating ln(CCP) or ln(1-CCP)
// - To avoid this problem, we replace 0s with "myzero" and
//    1s with "1-myzero"

myzero = 1e-6;
npest_ccp_0 = (npest_ccp_0.<=0).*myzero
            + (npest_ccp_0.>myzero).*(npest_ccp_0.<(1-myzero)).* npest_ccp_0
            + (npest_ccp_0.>=(1-myzero)).* (1-myzero) ;

npest_ccp_1 = (npest_ccp_1.<=0).*myzero
            + (npest_ccp_1.>myzero).*(npest_ccp_1.<(1-myzero)).* npest_ccp_1
            + (npest_ccp_1.>=(1-myzero)).* (1-myzero) ;

// Selection of realized CCPs in the sample
ccp_obs = (sim_y0.==0) .* npest_ccp_0[sim_s1] + (sim_y0.==1) .*
npest_ccp_1[sim_s1] ;
```

```
// Constructing W1[t]
w1obs = ln(ccp_obs) - ln(1-ccp_obs) ;

// Selection of realized transition probabiltiies ftran
ftran_obs = ftran[sim_s1,.] ;

// Constructing W2[t]
w2obs = delta * ftran_obs *(ln(1-npest_ccp_1) - ln(1-npest_ccp_0)) ;

// Constructing W[t]
wobs = w1obs + w2obs ;

// 8.2. OLS estimation of theta parameters
depvar = wobs ;
explan = ones(nobs,1)~valsize[sim_s1]~(1-sim_y0) ;

call ols(0, depvar, explan);

output off ;

end ;
```