```
new ;
closeall ;

//  ************************************************************************
//
//  incae_tutorial_01_solution_singleagent_entryexit.gss
//
//  Program for the solution of single-agent dynamic programming entry-exit model
//
//  by Victor Aguirregabiria
//
//  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_01_solution_singleagent_entryexit.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
// ------------------------------------------------
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);

closeall;

end ;
```