```
new ;
closeall ;

//  ************************************************************************
//
//  incae_tutorial_04_estimation_dynamic_game.gss
//
//  Program for the estimation of dynamic game of market entry-exit with 2 firms
//
//  Given parameters fixed by user, the program:
//      (a) computes an equilibrium (Markov Perfect equilibrium) of the model
//      (b) generates an artifitial / simulated dataset from that equilibrium
//      (c) uses the simulated as actual data and estimates the parameters of the
model
//
//  The estimation methods are:
//      (a) Two-step Pseudo Maximum Likelihood
//      (b) Iterated Pseudo Maximum Likelihood (NPL) (Full solution method)
//
//  by Victor Aguirregabiria
//
//  ************************************************************************

library pgraph ;

// --------------
//  OUTPUT FILE
// --------------
wdir = "C:\\Users\\victo\\Dropbox\\documents\\COURSES\\2022_INCAE\\TUTORIALS\\" ;
// Working directory
buff = changedir(wdir) ;
fileout  = "incae_tutorial_03_solution_dynamic_game.out" ; // Name of output file
output file = ^fileout reset ;
format /mb1 /ros 16,4 ;

// ----------------------------------------------------------------
//  Specification of Profit Function
//
//  i = index for firm. i belongs to {1, 2}
//  Firms' actions = a_1 and a_2
//
//      a_i = 0 ----> Firm i is not active in market
//      a_i = 1 ----> Firm i is active in market
//
// ----------------------------------------------------------------
//
//  The game is dynamic because there is a sunk cost of entry
//
//  The only state variables of this game are the indicators of whether
//  the firms were active or not in the market at previous period.
```

```
//  These are payoff relevant state variables because they determine
//  whether a firm has to pay an entry cost to operate in the market.
//
//          x_i = a_i[t-1]
//
//          x_i = 0 ----> Firm i was not active in market at previous period
//              = 1 ----> Firm i was active in market at previous period
//
// ------------------------------------------------------------------
//
//  Parameters:
//      ecost =   Entry cost
//      pi_mon =   Operating profit of a monopolist
//      pi_duo =   Operating profit of a duopolist
//
// ------------------------------------------------------------------
//
//  Profit function for firm 1 (the symmetric for firm 2)
//
//      Profit_1(a_1 = 0) = eps_1(0)
//
//      Profit_1(a_1 = 1) = (1-a_2) * pi_mon + a_2 * pi_duo
//
//                          - (1-a_1[t-1]) * ecost + eps_1(1)
//
//  where eps_1(0) and eps_1(1) are private information shocks that are
//  i.i.d. over time and over players, and independent of each other,
//  with Standard Normal distribution.
//
// ------------------------------------------------------------------

// ------------------------------------------------
//  1. Values of parameters and other constants
// ------------------------------------------------
nobs = 1000 ;   // Number of observations (local markets) in simulated dataset
npliter = 20 ;  // Maximum Number of NPL iterations for estimation

ecost  = 1.0 ;    // Entry cost
pi_mon = 1.0 ;     // Monopoly  profit
pi_duo = 0.4 ;     // Duopoly profit

dfact   = 0.9 ;    // Discount factor

nplayer = 2 ;       // Number of players

trueparam = (ecost | pi_mon | pi_duo) ;  // vector with true values of parameters
namesb = "Entry Cost"
       | "Monopoly Prof"
       | "Duopoly Prof" ; // Vector with names of parameters
kparam  = rows(trueparam) ;   //  Number of parameters to estimate
```

```
// 'vstate' is a matrix with all the possible values of
//  the state variables of this dynamic game.
//  That is, a matrix with all the possible values of x_1 and x_2
//
//  - Each row represents a value for the state variables
//  - Column 1 contains the values of x_1, and column 2 the values of x_2

vstate = (0 ~ 0)
       | (0 ~ 1)
       | (1 ~ 0)
       | (1 ~ 1) ;

nstate = rows(vstate) ; // Number of possible values of the vector of state
variables

//
------------------------------------------------------------------------------
------
//  2. PROCEDURE FOR EVALUATING BEST-RESPONSE MAPPING
//
------------------------------------------------------------------------------
------
//
//  Input of the procedure:
//      prob0 = (nstate x 2) matrix with the CCPs of each player
//
//      Each row corresponds to a state, each column corresponds to the CCP of a
player
//
//  Output of the procedure:
//      newprob = (nstate x 2) matrix with the best response CCPs of each player
//
//      The i-th column is the vector of CCPs which are player i's best response
to prob0
//
//
------------------------------------------------------------------------------
------


proc (1) = equilmap(prob0);
  local ns, newprob, profit1_a0, profit1_a1, profit2_a0, profit2_a1,
        iptran1_a0, iptran1_a1, iptran2_a0, iptran2_a1,
        eprofit1, eprofit2, ftran, value1, value2, vtilda1, vtilda2 ;
  ns = rows(prob0) ;    // number of states
  newprob = prob0 ;

  // ------------------------------
  // a. Vectors of expected profits
```

```
  // ------------------------------

  // Expected profit of firms 1 and 2 if own action is 0
  profit1_a0 = zeros(ns,1) ;
  profit2_a0 = zeros(ns,1) ;

  // Expected profit of firms 1 and 2 if own action is 1
  profit1_a1 = (1-prob0[.,2]).*pi_mon + prob0[.,2].*pi_duo -
ecost.*(1-vstate[.,1]) ;
  profit2_a1 = (1-prob0[.,1]).*pi_mon + prob0[.,1].*pi_duo -
ecost.*(1-vstate[.,2]) ;

  // Ex-ante expected profit of firms 1 and before knowing the own epsilons
  // Assumption about distribution of epsilon: Standard Normal
  eprofit1 = (1-prob0[.,1]).*profit1_a0 + prob0[.,1].*profit1_a1
           + pdfn(cdfni(prob0[.,1])) ;

  eprofit2 = (1-prob0[.,2]).*profit2_a0 + prob0[.,2].*profit2_a1
           + pdfn(cdfni(prob0[.,2])) ;

  // ------------------------------------------------------
  //  b. Transition probabilities of the state variables
  // ------------------------------------------------------
  //
  //  - (nstate x nstate) matrices of transition probabilities of
  //    state variables
  //
  //  - In these transition matrices:
  //        - Rows represent current value of state variables
  //        - Columns represent next period value of state variables
  //            (the elements of each row should sum to 1)
  //
  //  - Remember: vstate = (0~0) | (0~1) | (1~0) | (1~1)
  //
  //  - Since the only state variables are the lagged values
  //    of firms' actions, all what we need to obtain the
  //    transition probabilities is the choice probabilities
  //
  //  - Transition matrices from the point of view of each firm:
  //    Each firm knows his current action but not the current action
  //    of his competitor
  //
  // ------------------------------------------------------

  // Trasition matrix of firm 1 if current own action = 0
  //  Remember: vstate = (0~0)     | (0~1)          | (1~0)       | (1~1)
  iptran1_a0 =      (1-prob0[.,2]) ~ prob0[.,2]  ~ zeros(ns,1)~ zeros(ns,1) ;

  // Trasition matrix of firm 1 if current own action = 1
  //  Remember: vstate = (0~0)     | (0~1)          | (1~0)       | (1~1)
```

```
  iptran1_a1 =      zeros(ns,1)    ~ zeros(ns,1) ~ (1-prob0[.,2]) ~ prob0[.,2] ;

  // Trasition matrix of firm 2 if current own action = 0
  //  Remember: vstate = (0~0)     | (0~1)          | (1~0)       | (1~1)
  iptran2_a0 =      (1-prob0[.,1]) ~ zeros(ns,1)    ~ prob0[.,1] ~ zeros(ns,1) ;

  // Trasition matrix of firm 2 if current own action = 1
  //  Remember: vstate = (0~0)     | (0~1)          | (1~0)       | (1~1)
  iptran2_a1 =      zeros(ns,1)    ~ (1-prob0[.,1]) ~ zeros(ns,1)~ prob0[.,1] ;

  // Ex-ante transition probability matrix
  //    Not conditional on own action
  //  Remember: vstate = (0~0) | (0~1) | (1~0) | (1~1)

   ftran = (1-prob0[.,1]).*(1-prob0[.,2])
         ~ (1-prob0[.,1]).*prob0[.,2]
         ~ prob0[.,1].*(1-prob0[.,2])
         ~ prob0[.,1].*prob0[.,2] ;

  //  -------------------------------------------------------
  //  c. Calculating present values for each firm and state
  //        Given that firms' believe that they will behave
  //        in the future according to the probabilities prob0
  //  -------------------------------------------------------
  ftran = inv(eye(ns)-dfact*ftran) ;    // 'Discounting' matrix
  value1 = ftran * eprofit1 ;   // Present values for firm 1
  value2 = ftran * eprofit2 ;   // Present values for firm 2

  // Threshold value for firm 1:
  //    Value of action 1 minus value of action 0
  vtilda1 = (profit1_a1 + dfact * iptran1_a1 * value1)
          - (profit1_a0 + dfact * iptran1_a0 * value1) ;

  // Threshold value for firm 2:
  //    Value of action 1 minus value of action 0
  vtilda2 = (profit2_a1 + dfact * iptran2_a1 * value2)
          - (profit2_a0 + dfact * iptran2_a0 * value2) ;

  //  -------------------------------------------------------
  //  d. Best response probabilities
  //  -------------------------------------------------------
  newprob = cdfn(vtilda1) ~ cdfn(vtilda2) ;
  retp(newprob) ;
endp ;
// -------------- End procedure equimap -----------------


// ------------------------------------------------------------------------
//  3. COMPUTING AN EQUILIBRIUM (MPE)
// ------------------------------------------------------------------------
```

```
//
//  Here the algorithm for computing an equilibrium
//  is fixed point iterations in the best response equilibrium mapping
//
// ------------------------------------------------------------------------

prob0 = rndu(nstate,nplayer) ; // Initial value for the vector of players' CCPs

cconv = 1e-12 ;  // Convergence constant
criter = 1000 ;  // Initial value for the convergence criterion
maxiter = 100 ;  // Maximum number of iterations to stop
                 // Remember, the equilibrium mapping is not a contraction
                 // such that fixed point iterations might not converge

iter=1 ;
do while (criter>cconv).and(iter<=maxiter) ;
  "         Best response mapping iteration  =";; iter ;
  "         Convergence criterion =";; criter ;
  "" ;
  prob1 = equilmap(prob0);
  criter = maxc(maxc(abs(prob1-prob0))) ;
  prob0 = prob1 ;
  iter=iter+1 ;
endo ;
pequil = prob0 ;

if (iter<maxiter) ;
  "-------------------------------------------------------------------------" ;
  "         CONVERGENCE ACHIEVED AFTER";; iter;; "BEST RESPONSE ITERATIONS";
  "-------------------------------------------------------------------------" ;
  "         EQUILIBRIUM PROBABILITIES" ;
  pequil ;
  "" ;
  "     Remember that" ;
  "" ;
  "     vstate  = (0 ~ 0)" ;
  "             | (0 ~ 1)" ;
  "             | (1 ~ 0)" ;
  "             | (1 ~ 1)" ;
  "-------------------------------------------------------------------------" ;
endif ;


// ------------------------------------------------------------------
// 4. COMPUTING THE STEADY-STATE DISTRIBUTION OF THE STATE VARIABLES
//      for THE COMPUTED EQUILIBRIUM
//
//      This is the distribution of "market structure" in steady state
// ------------------------------------------------------------------
```

```
// Transition probability matrix for the state variables
ftran = (1-pequil[.,1]).*(1-pequil[.,2])
      ~ (1-pequil[.,1]).*pequil[.,2]
      ~ pequil[.,1].*(1-pequil[.,2])
      ~ pequil[.,1].*pequil[.,2] ;

// By definition, the steady-state distribution 'psteady'
//  is such that:
//
//      psteady = Ftran * psteady
//
//  where Ftran is the transition probability matrix of the state vars
//
//   We can calculate psteady by fixed point iterations in the vector
//   system:  psteady = Ftran * psteady
// ------------------------------------------------------------------------

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

"------------------------------------------------------------------------" ;
"       Ergodic Distribution of the State Variables" ;
"" ;
psteady ;

"" ;
"------------------------------------------------------------------------" ;
"" ;
"       Remember that" ;
"" ;
"           vstate  =   (0 ~ 0)" ;
"                     | (0 ~ 1)" ;
"                     | (1 ~ 0)" ;
"                     | (1 ~ 1)" ;
"" ;
"------------------------------------------------------------------------" ;


// ------------------------------------------------------------------------
//  5. PROCEDURE TO SIMULATE DATA FROM THE COMPUTED EQUILIBRIUM
// ------------------------------------------------------------------------
//
```

```
//  Inputs of the procedure:
//
//      numobs  = Number of observations (markets) in the simulated dataset
//      pchoice = Vector of equilibrium CCPs
//      pste    = Vector with steady-state distribution of the statte variables
//      vecs    = Matrix with the values of the possible values of the state
variables
//                (in a two-player model of market entry-exit, we have that
//                vecs = (0 ~ 0) | (0 ~ 1) | (1 ~ 0) | (1 ~ 1)
//
//  Outputs of the procedure:
//      simul_a     =   (numobs x 2) matrix with observations at every market
//                      of actions of firms 1 and 2 at period t
//      simul_a_1   =   (numobs x 2) matrix with observations at every market
//                      of actions of firms 1 and 2 at period t-1
//
// ----------------------------------------------------------------

proc (2) = simdygam(numobs,pchoice,pste,vecs) ;
  local seed, nums, nump, pbuff0, pbuff1, uobs, simul_a, simul_a_1 ;
  nums = rows(vecs) ;
  nump = cols(pchoice) ;
  pbuff1 = cumsumc(pste) ;
  pbuff0 = cumsumc((0|pste[1:nums-1])) ;
  uobs = rndu(numobs,1) ;
  uobs = ((uobs.>=(pbuff0')).*(uobs.<=(pbuff1'))) * seqa(1,1,nums) ;
  simul_a_1 = vecs[uobs,.] ;
  simul_a = (rndu(numobs,nump).<=pchoice[uobs,.]) ;
  retp(simul_a,simul_a_1) ;
endp ;


// ------------------------
//  6. SIMULATING DATA
// ------------------------
{aobs , aobs_1} = simdygam(nobs,pequil,psteady,vstate) ;


// ---------------------------------------
//  7.  PROCEDURE for FREQUENCY ESTIMATOR
// ---------------------------------------
proc (1) = freqprob(yobs,xobs,xval) ;
  local numx, numq, prob1, t, selx, denom, numer ;
  numx = rows(xval) ;
  numq = cols(yobs) ;
  prob1 = zeros(numx,numq) ;
  t=1 ;
  do while t<=numx ;
    selx = prodc((xobs.==xval[t,.])') ;
    denom = sumc(selx) ;
```

```
    if (denom==0) ;
      prob1[t,.] = zeros(1,numq) ;
    else ;
      numer = sumc(selx.*yobs) ;
      prob1[t,.] = (numer')./denom ;
    endif ;
    t=t+1 ;
  endo ;
  retp(prob1) ;
endp ;

// ----------------------------------------
//  8.  FREQUENCY ESTIMATES OF CCPs
// ----------------------------------------
freq_est_prob = freqprob(aobs,aobs_1,vstate) ;

"" ;
"-------------------------------------------------------------------------" ;
"       Estimated Ergodic Distribution of State Variables" ;
"" ;
est_psteady = zeros(nstate,1) ;
est_psteady[1] = sumc(prodc((aobs_1.==vstate[1,.])'))/nobs ;
est_psteady[2] = sumc(prodc((aobs_1.==vstate[2,.])'))/nobs ;
est_psteady[3] = sumc(prodc((aobs_1.==vstate[3,.])'))/nobs ;
est_psteady[4] = sumc(prodc((aobs_1.==vstate[4,.])'))/nobs ;

est_psteady ;
"" ;
"       True Ergodic Distribution of State Variables" ;
psteady ;
"" ;
"-------------------------------------------------------------------------" ;
"" ;
"       Estimated Conditional Choice Probabilities" ;
freq_est_prob ;
"" ;
"       True Conditional Choice Probabilities" ;
pequil ;
"" ;
"-------------------------------------------------------------------------" ;


// ------------------------------------------------
//  9. PROCEDURE for CONSTRAINED PROBIT ESTIMATOR
// ------------------------------------------------

proc (3) = miprobit(ydum,x,rest,b0,out) ;
  local myzero, nobs, nparam, eps, namesb, iter, llike,
        criter, Fxb0, phixb0, lamdab0, dlogLb0,
        d2logLb0, b1, lamda0, lamda1, Avarb, sdb, tstat,
```

```
      numy1, numy0, logL0, LRI, pseudoR2, k ;
myzero = 1e-36 ;
nobs = rows(ydum) ;
nparam = cols(x) ;
eps = 1E-6 ;
namesb = seqa(1,1,nparam) ;
namesb = 0 $+ "b" $+ ftocv(namesb,2,0);
iter=1 ;
llike = 1000 ;
criter = 1000 ;
do while (criter>eps) ;
  if (out==1) ;
    "" ;
    "Iteration                = " iter ;
    "Log-Likelihood function  = " llike ;
    "Criterion                = " criter ;
    "" ;
  endif ;
  Fxb0 = cdfn(x*b0+rest) ;
  Fxb0 = Fxb0 + (myzero - Fxb0).*(Fxb0.<myzero)
              + (1-myzero - Fxb0).*(Fxb0.>(1-myzero));
  llike = ydum'*ln(Fxb0) + (1-ydum)'*ln(1-Fxb0) ;
  phixb0 = pdfn(x*b0+rest) ;
  lamdab0 = ydum.*(phixb0./Fxb0) + (1-ydum).*(-phixb0./(1-Fxb0)) ;
  dlogLb0 = x'*lamdab0 ;
  d2logLb0 = -((lamdab0.*(lamdab0 + x*b0 + rest)).*x)'*x ;
  b1 = b0 - inv(d2logLb0)*dlogLb0 ;
  criter = maxc(abs(b1-b0)) ;
  b0 = b1 ;
  iter = iter + 1 ;
endo ;
Fxb0 = cdfn(x*b0 + rest) ;
Fxb0 = Fxb0 + (myzero - Fxb0).*(Fxb0.<myzero)
            + (1-myzero - Fxb0).*(Fxb0.>(1-myzero));
llike = ydum'*ln(Fxb0) + (1-ydum)'*ln(1-Fxb0) ;
phixb0 = pdfn(x*b0 + rest) ;
lamda0 = -phixb0./(1-Fxb0) ;
lamda1 =  phixb0./Fxb0 ;
Avarb  = ((lamda0.*lamda1).*x)'*x ;
Avarb  = inv(-Avarb) ;
sdb    = sqrt(diag(Avarb)) ;
tstat  = b0./sdb ;
numy1  = sumc(ydum) ;
numy0  = nobs - numy1 ;
logL0  = numy1*ln(numy1) + numy0*ln(numy0) - nobs*ln(nobs) ;
LRI    = 1 - llike/logL0 ;
pseudoR2 = 1 - ( (ydum - Fxb0)'*(ydum - Fxb0) )/numy1 ;
if (out==1) ;
  "Number of Iterations     = " iter ;
  "Log-Likelihood function  = " llike ;
```

```
    "Likelihood Ratio Index   = " LRI ;
    "Pseudo-R2                = " pseudoR2 ;
    "" ;
    "-----------------------------------------------------------------";
    "       Parameter     Estimate        Standard        t-ratios";
    "                                     Errors" ;
    "-----------------------------------------------------------------";
    k=1;
    do while k<=nparam;
      print $namesb[k];;b0[k];;sdb[k];;tstat[k];
      k=k+1 ;
    endo;
    "-----------------------------------------------------------------";
  endif ;
  retp(b0,Avarb,llike) ;
endp ;


// ---------------------------------
// 10. PROCEDURE for NPL ITERATIONS
// ---------------------------------
proc (3) = npldygam(yobs,yobs_1,pchoice,disfact,mstate,b0,kiter,namesb);
  local myzero, nobs, nplayer, numx, kparam, best, varb,
        iter, cconv, criter, pchoice_0,  pchoice_1,
        indobs, u0, u1, e0, e1, ptran, inv_bf,
        iptran1_a0, iptran1_a1, iptran2_a0, iptran2_a1,
        umat1_a0, umat1_a1, umat2_a0, umat2_a1,
        zmat1, zmat2, emat1, emat2,
        zobs, eobs, tetaest, varest, likelihood,
        sdb, tstat, k ;

  // ---------------
  // Some constants
  // ---------------
  myzero = 1e-16 ;
  nobs = rows(yobs) ;
  nplayer = cols(yobs) ;
  numx = rows(pchoice) ;
  kparam = rows(b0) ;
  best = zeros(kparam,kiter) ;
  varb = zeros(kparam,kparam*kiter) ;
  yobs = yobs[.,1] | yobs[.,2] ;

  // ----------------------------------------
  // Vector with indexes for the observed state
  // ----------------------------------------
  indobs = 1.*prodc((yobs_1.==mstate[1,.])')
         + 2.*prodc((yobs_1.==mstate[2,.])')
         + 3.*prodc((yobs_1.==mstate[3,.])')
         + 4.*prodc((yobs_1.==mstate[4,.])') ;
```

```
  // -------------
  // NPL algorithm
  // -------------
  cconv = 1e-6 ;    // convergence constant
  criter = 1000 ;   //  convergence criterion
  pchoice_0 =  pchoice ;   // Initial vector of CCPs
  iter=1 ;
  do while (iter<=kiter).and(criter>cconv) ;
    // ---------------------------------------
    // a. Truncation of probabilities to avoid
    //    inverse Mill's ratio = +INF
    // ---------------------------------------
    pchoice_0[.,1] = (pchoice_0[.,1].<myzero).*myzero
                   + (pchoice_0[.,1].>(1-myzero)).*(1-myzero)
                   +
(pchoice_0[.,1].>=myzero).*(pchoice_0[.,1].<=(1-myzero)).*pchoice_0[.,1] ;
    pchoice_0[.,2] = (pchoice_0[.,2].<myzero).*myzero
                   + (pchoice_0[.,2].>(1-myzero)).*(1-myzero)
                   +
(pchoice_0[.,2].>=myzero).*(pchoice_0[.,2].<=(1-myzero)).*pchoice_0[.,2] ;

    // -------------------------------------
    // b. Matrix of transition probabilities
    // -------------------------------------
    ptran = (1-pchoice_0[.,1]).*(1-pchoice_0[.,2])
          ~ (1-pchoice_0[.,1]).*pchoice_0[.,2]
          ~ pchoice_0[.,1].*(1-pchoice_0[.,2])
          ~ pchoice_0[.,1].*pchoice_0[.,2] ;

    // --------------------
    // c. Inverse of I-b*F
    // --------------------
    inv_bf = inv(eye(numx)-disfact*ptran) ;

    // ------------------------------------
    // d. Matrices Pr(a[t] | a[t-1], ai[t])
    // ------------------------------------
    iptran1_a0 = (1-pchoice_0[.,2]) ~ pchoice_0[.,2]  ~ zeros(nstate,1)   ~
zeros(nstate,1) ;
    iptran1_a1 = zeros(nstate,1)    ~ zeros(nstate,1) ~ (1-pchoice_0[.,2])~
pchoice_0[.,2] ;
    iptran2_a0 = (1-pchoice_0[.,1]) ~ zeros(nstate,1) ~ pchoice_0[.,1]    ~
zeros(nstate,1) ;
    iptran2_a1 = zeros(nstate,1)    ~ (1-pchoice_0[.,1])~ zeros(nstate,1)   ~
pchoice_0[.,1] ;

    // ----------------------------------------
    // e. Construction of explanatory variables
    // ----------------------------------------
```

```
umat1_a0 = zeros(numx,kparam) ;
umat1_a1 = (-(1-mstate[.,1]))~(1-pchoice_0[.,2])~pchoice_0[.,2] ;
umat2_a0 = zeros(numx,kparam) ;
umat2_a1 = (-(1-mstate[.,2]))~(1-pchoice_0[.,1])~pchoice_0[.,1] ;

zmat1 = (1-pchoice_0[.,1]).*umat1_a0 + pchoice_0[.,1].*umat1_a1 ;
zmat1 = inv_bf * zmat1 ;
zmat1 = (umat1_a1 + disfact * iptran1_a1 * zmat1)
      - (umat1_a0 + disfact * iptran1_a0 * zmat1) ;

zmat2 = (1-pchoice_0[.,2]).*umat2_a0 + pchoice_0[.,2].*umat2_a1 ;
zmat2 = inv_bf * zmat2 ;
zmat2 = (umat2_a1 + disfact * iptran2_a1 * zmat2)
      - (umat2_a0 + disfact * iptran2_a0 * zmat2) ;

emat1 = pdfn(cdfni(pchoice_0[.,1])) ;
emat1 = inv_bf * emat1 ;
emat1 = (disfact * iptran1_a1 * emat1)
      - (disfact * iptran1_a0 * emat1) ;

emat2 = pdfn(cdfni(pchoice_0[.,2])) ;
emat2 = inv_bf * emat2 ;
emat2 = (disfact * iptran2_a1 * emat2)
      - (disfact * iptran2_a0 * emat2) ;

zobs = zmat1[indobs,.] | zmat2[indobs,.] ;
eobs = emat1[indobs,.] | emat2[indobs,.] ;

// ----------------------------------------
// f. Pseudo Maximum Likelihood Estimation
// ----------------------------------------
{tetaest,varest,likelihood} = miprobit(yobs,zobs,eobs,zeros(kparam,1),0) ;
best[.,iter] = tetaest ;
varb[.,(iter-1)*kparam+1:iter*kparam] = varest ;
sdb    = sqrt(diag(varest)) ;
tstat  = tetaest./sdb ;

// ------------------------
// g. Updating probabilities
// ------------------------
pchoice_1 = cdfn(zmat1*tetaest +emat1)
          ~ cdfn(zmat2*tetaest +emat2) ;

// ----------------------------
// h. Checking for Convergence
// ----------------------------
if (iter==1) ;
  criter = 1000 ;
elseif (iter>1) ; ;
  criter = maxc(maxc(abs(best[.,iter]-best[.,iter-1]))) ;
```

```
    endif ;

    // ------------------------------------------------------------
    // i. Presenting estimaton results from current NPL iteration
    // ------------------------------------------------------------

    "" ;
    "-------------------------------------------------------------------------" ;
    "       NPL ITERATION NUMBER        =" ;; iter;
    "       NPL Convergence Criterion   =" ;; criter ;
    "" ;
    "       Estimation Results from Current Iteration";
    "-------------------------------------------------------------------------" ;
    "       Log-Likelihood function  = " likelihood ;
    "" ;
    "-------------------------------------------------------------------";
    "       Parameter           Estimate        Standard        t-ratios";
    "                                           Errors" ;
    "-------------------------------------------------------------------";
    "" ;
    k=1 ;
    do while k<=kparam ;
      print $namesb[k];; tetaest[k];;sdb[k];;tstat[k];
      k=k+1 ;
    endo;
    "" ;
    "-------------------------------------------------------------------";
    "" ;

    pchoice_0 = pchoice_1 ;
    iter=iter+1 ;
  endo ;

    "" ;
    "-------------------------------------------------------------------------" ;
    "-------------------------------------------------------------------------" ;
    "   NPL ESTIMATION" ;
    "       CONVERGENCE ACHIEVED AFTER";; (iter-1);; "NPL ITERATIONS";
    "-------------------------------------------------------------------------" ;
    "-------------------------------------------------------------------------" ;

  retp(best,varb,likelihood) ;
endp ;

// ---------------------------
//  11. NPL Estimation
// ---------------------------

"" ;
"*************************************************************************" ;
```

```
"*************************************************************************" ;
"   ESTIMATION OF THE STRUCTURAL PARAMETERS OF THE DYNAMIC GAME" ;
"   BASED ON SIMULATED DATA FROM THE COMPUTED MPE" ;
"*************************************************************************" ;
"*************************************************************************" ;
"" ;
"-------------------------------------------------------------------------" ;
theta0 = zeros(kparam,1) ;

{best1,varb,like1}
  = npldygam(aobs,aobs_1,freq_est_prob,dfact,vstate,theta0,npliter,namesb) ;


"" ;
"-----------------------------------------------------------------";
"       True value of paramaters:" ;
"" ;
"           Entry Cost      =";; ecost ;
"           Monopoly Profit =";; pi_mon ;
"           Duopoly Profit  =";; pi_duo ;
"" ;
"-----------------------------------------------------------------";
output off ;

end ;
```