Annotated MATLAB Translation of the Aguirregabiria and Mira (2007) Replication Code

Jason R. Blevins and Minhae Kim

The Ohio State University, Department of Economics

November 12, 2019

Introduction

This is an annotated MATLAB translation of the Monte Carlo source code for the experiments in Section 4 of Aguirregabiria and Mira (2007). The original source code[1] was written in Gauss in by Victor Aguirregabiria. This is close to a direct translation of the original Gauss code to MATLAB, with annotations added, written by Jason Blevins and Minhae Kim. The complete source and results can be found at `https://jblevins.org/research/am_2007`.

This program solves, simulates, and estimates a discrete time dynamic discrete choice entry game among $N = 5$ heterogeneous firms. Many comments from the original source code are preserved below.

This code is also the basis for the Monte Carlo experiments in Blevins and Kim (2019), which applies the NPL estimator to continuous-time dynamic discrete games.

Model

Main features of the model:

- Dynamic game of firm entry and exit in a single market.
- Firms are indexed by $i = 1, 2, 3, 4, 5$ and t is the time index.
- The binary decision variable is a_{it} defined as $a_{it} = 1$ when firm i is active in the market in period t and $a_{it} = 0$ otherwise.

[1] Source code dated May 2005, with comments and explanations added in August 2011, may be obtained at `http://individual.utoronto.ca/vaguirre/software/`. See Aguirregabiria (2009) for detailed discussion of the code.

- The game is dynamic because firms must pay an entry cost θ_{EC} to enter the market.
- The state variables of the game in period t are:

  1. Five binary variables indicating which firms were active in the previous period $(a_{1,t-1}, \ldots, a_{5,t-1})$,
  2. Market size (s_t),
  3. Private choice-specific information shocks for each firm (ε_{it}).

  These states are payoff-relevant because they determine whether a firm has to pay an entry cost to operate in the market. The full state vector for firm i is (x_t, ε_{it}) where $x_t = (s_t, a_{1,t-1}, a_{2,t-1}, a_{3,t-1}, a_{4,t-1}, a_{5,t-1})$ and $\varepsilon_{it} = (\varepsilon_{it}(0), \varepsilon_{it}(1))$ and where $\varepsilon_{it}(0)$ and $\varepsilon_{it}(1)$ are known only to firm i.
- The profit function for an inactive firm i $(a_{it} = 0)$ is:

$$\Pi_{it}(0) = \varepsilon_{it}(0).$$

  For an active firm i $(a_{it} = 1)$, the profit function is:

$$\Pi_{it}(1) = \theta_{\text{FC},i} - \theta_{\text{EC}}(1 - a_{i,t-1}) + \theta_{\text{RS}} s_t - \theta_{\text{RN}} \ln(N_{-it} + 1) + \varepsilon_{it}(1),$$

  where N_{-it} denotes the number of rival firms (firms other than firm i) operating in the market in period t and $\theta_{\text{FC},i}$, θ_{EC}, θ_{RS} and θ_{RN} are parameters.
- The private information shocks $\varepsilon_{it}(0)$ and $\varepsilon_{it}(1)$ are independent and identically distributed across time, markets, and players, are independent of each other, and follow the standard type I extreme value distribution.

Structure of the Program

The code for the main control program proceeds in 8 steps and is presented first, followed by the other subprograms called by the control program:

1 Main control program

  1 Selection of the Monte Carlo experiment to implement
  2 Values of parameters and other constants
  3 Computing a Markov perfect equilibrium of the dynamic game
  4 Simulating data from the equilibrium to obtain descriptive statistics
  5 Checking for the consistency of the estimators (estimation with a very large sample)

1. Main control program

1.1. Selection of the Monte Carlo Experiment to Implement

This program implements six different experiments. The variable `selexper` stores a value from 1 to 6 that represents the index of the Monte Carlo experiment to implement. A run of this program implements one experiment.

```
numexp = 6;      % Total number of Monte Carlo experiments
selexper = 1;    % Select a Monte Carlo experiment to run (1 to numexp)
```

This program creates several output files which are stored in the current working directory. The names of these output files can be customized below.

```
% Output file
fileout     = char(sprintf("am_2007_%d.log", selexper));
% Estimates for initial CCPs = frequency estimator
nfile_bNP   = char(sprintf("bnp_exper_%d", selexper));
% Estimates for initial CCPs = logit estimation
nfile_bSP   = char(sprintf("bsp_exper_%d", selexper));
% Estimates for initial CCPs = random U(0,1)
nfile_bR    = char(sprintf("br_exper_%d", selexper));
% Estimates for initial CCPs = true CCPs
nfile_btrue = char(sprintf("btrue_exper_%d", selexper));
```

Next we open the log file for saving output.

```
diary(fileout);
```

1.2. Values of Parameters and Other Constants

First, we define several constants to specify the dimension of the problem. These parameters are the same across the experiments including the number of local markets, $M = 400$, the number of time periods, $T = 1$, the number of players, $N = 5$, and the number of Monte Carlo simulations, 1000.

```
nobs = 400;     % Number of markets (observations)
nrepli = 1000;  % Number of Monte carlo replications
nplayer = 5;    % Number of players
```

Next, we define the parameters used for each experiment. Each of the following variables is a matrix where the rows correspond to the six individual experiments and, for variables where firms are heterogeneous, the columns correspond to the number of players.

Firms across all experiments share the same discount factor, 0.95, and firms $i = 1, \ldots, 5$ have the same fixed costs in all experiments

$$\theta_{\text{FC},1} = 1.9,$$
$$\theta_{\text{FC},2} = 1.8,$$
$$\theta_{\text{FC},3} = 1.7,$$
$$\theta_{\text{FC},4} = 1.6,$$
$$\theta_{\text{FC},5} = 1.5.$$

Firm 5 has the lowest fixed cost of operation and is therefore the most efficient. The coefficient on market size, θ_{RS}, and the standard deviation of the choice-specific shocks are held fixed across experiments.

```
theta_fc = zeros(numexp, nplayer);
theta_fc(:, 1) = -1.9;   % Fixed cost for firm 1 in all experiments
theta_fc(:, 2) = -1.8;   % Fixed cost for firm 2 in all experiments
theta_fc(:, 3) = -1.7;   % Fixed cost for firm 3 in all experiments
theta_fc(:, 4) = -1.6;   % Fixed cost for firm 4 in all experiments
theta_fc(:, 5) = -1.5;   % Fixed cost for firm 5 in all experiments
```

```
theta_rs = 1.0 * ones(numexp, 1); % theta_rs for each experiment
disfact = 0.95 * ones(numexp, 1); % discount factor for each experiment
sigmaeps = 1 * ones(numexp, 1);   % std. dev. epsilon for each experiment
```

The market size process is the same across all six experiments. The variable has five points of support $s_t \in \{1, 2, 3, 4, 5\}$ represented by the column vector `sval` in the source. The the state transition probability matrix for s_t is

$$\begin{pmatrix} 0.8 & 0.2 & 0.0 & 0.0 & 0.0 \\ 0.2 & 0.6 & 0.2 & 0.0 & 0.0 \\ 0.0 & 0.2 & 0.6 & 0.2 & 0.0 \\ 0.0 & 0.0 & 0.2 & 0.6 & 0.2 \\ 0.0 & 0.0 & 0.0 & 0.2 & 0.8 \end{pmatrix}.$$

```
% Points of support and transition probability of state variable s[t], market size
sval = [ 1:1:5 ]';  % Support of market size
numsval = size(sval, 1);  % Number of possible market sizes
nstate = numsval * (2^nplayer);    % Number of points in the state space
ptrans = [ 0.8, 0.2, 0.0, 0.0, 0.0 ;
           0.2, 0.6, 0.2, 0.0, 0.0 ;
           0.0, 0.2, 0.6, 0.2, 0.0 ;
           0.0, 0.0, 0.2, 0.6, 0.2 ;
           0.0, 0.0, 0.0, 0.2, 0.8 ];
```

The parameters that differ across the six experiments are the competitive effect, θ_{RN}, and the cost of entry, θ_{EC}:

1. Experiment 1: $\theta_{\text{EC}} = 1.0$ and $\theta_{\text{RN}} = 0.0$,
2. Experiment 2: $\theta_{\text{EC}} = 1.0$ and $\theta_{\text{RN}} = 1.0$,
3. Experiment 3: $\theta_{\text{EC}} = 1.0$ and $\theta_{\text{RN}} = 2.0$,
4. Experiment 4: $\theta_{\text{EC}} = 0.0$ and $\theta_{\text{RN}} = 1.0$,
5. Experiment 5: $\theta_{\text{EC}} = 2.0$ and $\theta_{\text{RN}} = 1.0$,
6. Experiment 6: $\theta_{\text{EC}} = 4.0$ and $\theta_{\text{RN}} = 1.0$.

```
theta_rn = [ 0.0; 1.0; 2.0; 1.0; 1.0; 1.0 ]; % Values of theta_rn for each experiment
theta_ec = [ 1.0; 1.0; 1.0; 0.0; 2.0; 4.0 ]; % Values of theta_ec for each experiment
```

Finally, given the value of `selexper` defined above we collect and keep only the parameters for the selected experiment.

```
% Selecting the parameters for the experiment
theta_fc = theta_fc(selexper, :)';
theta_rs = theta_rs(selexper);
disfact = disfact(selexper);
sigmaeps = sigmaeps(selexper);
theta_ec = theta_ec(selexper);
theta_rn = theta_rn(selexper);

% Vector with true values of parameters
trueparam = [ theta_fc; theta_rs; theta_rn; theta_ec; disfact; sigmaeps ];
```

For reporting purposes, we define a vector of strings containing the names of the individual parameters.

```
% Vector with names of parameters of profit function
namesb = [ 'FC_1'; 'FC_2'; 'FC_3'; 'FC_4'; 'FC_5'; '  RS'; '  RN'; '  EC' ];
```

Define a structure to encapsulate the parameter values and settings:

```
% Structure for storing parameters and settings
param.theta_fc = theta_fc;
param.theta_rs = theta_rs;
param.theta_rn = theta_rn;
param.theta_ec = theta_ec;
param.disfact = disfact;
param.sigmaeps = sigmaeps;
param.sval = sval;
param.ptrans = ptrans;
param.verbose = 1;
```

Note that using this structure is a deviation from the original Gauss code, which passed these values using global variables. The original code also did not have the `verbose` flag, which was added to control the amount of output.

Finally, we set the seed of the internal pseudo-random number generator so that our results will be reproducible.

```
% Seed for (pseudo) random number generation
rand('seed', 20150403);
```

1.3. Computing a Markov Perfect Equilibrium of the Dynamic Game

```
maxiter = 200;    % Maximum number of Policy iterations
prob0 = 0.5 * rand(nstate, nplayer);
[ pequil, psteady, vstate, dconv ] = mpeprob(prob0, maxiter, param);
```

1.4. Simulating Data from the Equilibrium

Here we call the `simdygam` function to simulate 50,000 observations from the MPE to obtain descriptive statistics on the dynamics of market structure.

```
nobsfordes = 50000;
[ aobs, aobs_1, sobs ] = simdygam(nobsfordes, pequil, psteady, vstate);
```

After simulating a large sample, calculate and report several useful descriptive statistics. First print a heading.

```
fprintf("\n");
fprintf("*********************************************************************************\n");
fprintf("   DESCRIPTIVE STATISTICS FROM THE EQUILIBRIUM\n");
fprintf("   BASED ON %d OBSERVATIONS\n", nobsfordes);
fprintf("\n");
fprintf("   TABLE 2 OF THE PAPER AGUIRREGABIRIA AND MIRA (2007)\n");
fprintf("*********************************************************************************\n");
fprintf("\n");
```

Now calculate the summary statistics:

```
nf = sum(aobs')';      % Number of active firms in the market at t
nf_1 = sum(aobs_1')';  % Number of active firms in the market at t-1

% Regression of (number of firms t) on (number of firms t-1)
[ b, sigma ] = mvregress([ ones(nobsfordes, 1), nf_1 ], nf);
bareg_nf = b(2);  % Estimate of autorregressive parameter
```

```
entries = sum((aobs.*(1-aobs_1))')';   % Number of new entrants at t
exits = sum(((1-aobs).*aobs_1)')';     % Number of firm exits at t
excess = mean(entries+exits-abs(entries-exits))'; % Excess turnover
buff = corr([ entries, exits ]);
corr_ent_exit = buff(1,2); % Correlation entries and exits
freq_active = mean(aobs)'; % Frequencies of being active
```

Note that in GNU Octave one can use `ols` above instead of MATLAB's `mvregress` or `fitlm`.

```
% [ b, sigma, rsq ] = ols(nf, [ ones(nobsfordes, 1), nf_1 ]);
```

Format and print the summary statistics:

```
fprintf('\n');
fprintf('------------------------------------------------------------------------------\n');
fprintf('       (1)    Average number of active firms   = %12.4f\n', mean(nf));
fprintf('------------------------------------------------------------------------------\n');
fprintf('       (2)    Std. Dev. number of firms        = %12.4f\n', std(nf));
fprintf('------------------------------------------------------------------------------\n');
fprintf('       (3)    Regression N[t] on N[t-1]        = %12.4f\n', bareg_nf);
fprintf('------------------------------------------------------------------------------\n');
fprintf('       (4)    Average number of entrants       = %12.4f\n', mean(entries)');
fprintf('------------------------------------------------------------------------------\n');
fprintf('       (5)    Average number of exits          = %12.4f\n', mean(exits)');
fprintf('------------------------------------------------------------------------------\n');
fprintf('       (6)    Excess turnover (in # of firms)  = %12.4f\n', excess);
fprintf('------------------------------------------------------------------------------\n');
fprintf('       (7)    Correlation entries and exits    = %12.4f\n', corr_ent_exit);
fprintf('------------------------------------------------------------------------------\n');
fprintf('       (8)    Probability of being active      =\n');
disp(freq_active)
fprintf('------------------------------------------------------------------------------\n');
fprintf('\n');
```

Note: In the original Gauss source, the internal procedures appear here. For MATLAB compatibility, these have been moved to the end of the file.

1.5. Checking for Consistency of the Estimators

To check for consistency of the estimators (or for possible programming errors) Aguirregabiria and Mira (2007) estimated the model using each of the estimators using a large

sample of 400,000 markets. In all the experiments, and for each considered estimation method, the estimates were equal to the true value up to the 4th decimal digit.

In this version of the program code, this has been omitted to save memory requirements and CPU time.

The user interested in checking for consistency can do it by using this program code with the following selections in Part 1:

```
%          nobs = 400000 ;    % Number of markets (observations)
%          nrepli = 1 ;       % Number of Monte carlo replications
```

1.6. Monte Carlo Experiment

First, we print a heading that indicates which Monte Carlo experiment was selected.

```
fprintf('\n');
fprintf('*******************************************************************************\n');
fprintf('       MONTE CARLO EXPERIMENT #%d\n', selexper);
fprintf('*******************************************************************************\n');
fprintf('\n');
```

```
kparam = size(trueparam, 1) - 2; %  Number of parameters to estimate
npliter = 20;                    %  Maximum number of NPL iterations
param.verbose = 0;

% Matrix that stores NPL fixed point estimates (for each replication and NPL iteration)
% when we initialize the NPL algorithm with nonparametric frequency estimates of CCPs
bmatNP = zeros(kparam, nrepli*npliter);

% Matrix that stores NPL fixed point estimates (for each replication and NPL iteration)
% when we initialize the NPL algorithm with logit estimates of CCPs
bmatSP = zeros(kparam, nrepli*npliter);

% Matrix that stores NPL fixed point estimates (for each replication and NPL iteration)
% when we initialize the NPL algorithm with U(0,1) random draws for the CCPs
bmatR = zeros(kparam, nrepli*npliter);

% Matrix that stores NPL fixed point estimates (for each replication and NPL iteration)
% when we initialize the NPL algorithm with the true CCPs
btruemat = zeros(kparam, nrepli);

% We set the counter 'redraws' to zero.
```

```
% Note: When there are multicollinearity problems in a Monte Carlo
% sample we ignore that sample and take a new one. We want to check
% for the number of times we have to make these re-draws and this is
% why we use the counter 'redraws'
redraws = 0;

for draw = 1:nrepli
  fprintf('     Replication = %d\n', draw);
  fprintf('         (a)     Simulations of x''s and a''s\n');
  flag = 1;
  while (flag==1)
    [ aobs, aobs_1, sobs ] = simdygam(nobs, pequil, psteady, vstate);
    check = sum(sum([ aobs, aobs_1 ]) == zeros(1, 2*nplayer));
    check = check + sum(sum([ aobs, aobs_1 ]) == (nobs .* ones(1, 2*nplayer)));
    if (check > 0)
        flag = 1;
    elseif (check == 0)
        flag = 0;
    end
    redraws = redraws + flag; % Counts the number re-drawings
  end

  fprintf('         (b.1)   Estimation of initial CCPs (Non-Parametric)\n');
  prob0NP = freqprob(aobs, [ sobs, aobs_1 ], vstate);

  fprintf('         (b.2)   NPL algorithm using frequency estimates as initial CCPs\n');
  [ best, varb ] = npldygam(aobs, sobs, aobs_1, sval, ptrans, prob0NP, disfact, npliter);
  bmatNP(:,(draw-1)*npliter+1:draw*npliter) = best;

  fprintf('         (c.1)   Estimation of initial CCPs (Semi-Parametric: Logit)\n');
  % Construct dependent (aobsSP) and explanatory variables (xobsSP)
  aobsSP = reshape(aobs', nobs*nplayer, 1);
  alphai = kron(ones(nobs,1), eye(nplayer));
  xobsSP = kron(sobs, ones(nplayer,1));
  nfirms_1 = kron(sum(aobs_1, 2), ones(nplayer,1));
  aobs_1SP = reshape(aobs_1', nobs*nplayer, 1);
  xobsSP = [ alphai, xobsSP, aobs_1SP, nfirms_1 ];
  % Logit estimation
  [ best, varest ] = milogit(aobsSP, xobsSP);
  % Construct probabilities
  vstateSP = [ ones(size(vstate,1), nplayer), vstate, ...
               sum(vstate(:,2:nplayer+1)')' ];
  best = [ diag(best(1:nplayer)) ; ...
           ones(1,5) * best(nplayer+1); ...
           eye(nplayer) * best(nplayer+2); ...
           ones(1,5) * best(nplayer+3) ];
```

```
  prob0SP = 1 ./ (1+exp(-vstateSP*best));

  fprintf('         (c.2)   NPL algorithm using Logit estimates as initial CCPs\n');
  [ best, varb ] = npldygam(aobs, sobs, aobs_1, sval, ptrans, prob0SP, disfact, npliter);
  bmatSP(:,(draw-1)*npliter+1:draw*npliter) = best;

  fprintf('         (d.1)   Estimation of initial CCPs (Completely Random)\n');
  prob0R = rand(size(vstate,1), nplayer);

  fprintf('         (d.2)   NPL algorithm using U(0,1) random draws as initial CCPs\n');
  [ best, varb ] = npldygam(aobs, sobs, aobs_1, sval, ptrans, prob0R, disfact, npliter);
  bmatR(:,(draw-1)*npliter+1:draw*npliter) = best;

  fprintf('         (e)     NPL algorithm using true values as initial CCPs\n');
  [ best, varb ] = npldygam(aobs, sobs, aobs_1, sval, ptrans, pequil, disfact, 1);
  btruemat(:,draw) = best;
end

fprintf('           Number of Re-drawings due to Multicollinearity = %d\n',  redraws);
```

1.7. Saving Results of the Monte Carlo Experiment

Save the results of the Monte Carlo experiments to the filenames stated in Part 1 above:

```
save(nfile_bNP, 'bmatNP');
save(nfile_bSP, 'bmatSP');
save(nfile_bR, 'bmatR');
save(nfile_btrue, 'btruemat');
```

1.8. Statistics from the Monte Carlo Experiments

This section reports the results in Tables 4 and 5 of Aguirregabiria and Mira (2007).[2]

```
bmatNP = reshape(bmatNP, [kparam*npliter, nrepli])';
bmatSP = reshape(bmatSP, [kparam*npliter, nrepli])';
bmatR = reshape(bmatR, [kparam*npliter, nrepli])';
btruemat = btruemat';
```

[2] We note that there is a mistake in the original Gauss code when reporting the "2step-Random" results (starting on line 1364). The original code reports results from `mean_bmatSP`, `median_bmatSP`, and `se_bmatSP` (Logit), instead of `mean_bmatR`, `median_bmatR`, and `se_bmatR` (Random).

```
% Empirical Means
mean_bmatNP = mean(bmatNP);
mean_bmatNP = reshape(mean_bmatNP,[kparam,npliter]);
mean_bmatSP = mean(bmatSP);
mean_bmatSP = reshape(mean_bmatSP,[kparam,npliter]);
mean_bmatR = mean(bmatR);
mean_bmatR = reshape(mean_bmatR,[kparam,npliter]);
mean_bmattrue = mean(btruemat);

% Empirical Medians
median_bmatNP = median(bmatNP);
median_bmatNP = reshape(median_bmatNP,[kparam,npliter]);
median_bmatSP = median(bmatSP);
median_bmatSP = reshape(median_bmatSP,[kparam,npliter]);
median_bmatR = median(bmatR);
median_bmatR = reshape(median_bmatR,[kparam,npliter]);
median_bmattrue = median(btruemat);

% Empirical Standard Errors
se_bmatNP = std(bmatNP);
se_bmatNP = reshape(se_bmatNP,[kparam,npliter]);
se_bmatSP = std(bmatSP);
se_bmatSP = reshape(se_bmatSP,[kparam,npliter]);
se_bmatR = std(bmatR);
se_bmatR = reshape(se_bmatR,[kparam,npliter]);
se_bmattrue = std(btruemat);

fprintf('\n');
fprintf('*************************************************************************\n');
fprintf('   MONTE CARLO EXPERIMENT # %d\n', selexper);
fprintf('   EMPIRICAL MEANS AND STANDARD ERRORS\n');
fprintf('\n');
fprintf('   TABLE 4 OF THE PAPER AGUIRREGABIRIA AND MIRA (2007)\n');
fprintf('*************************************************************************\n');
fprintf('\n');
fprintf('-------------------------------------------------------------------\n');
fprintf('                     theta_fc_1        theta_rs        theta_rn        theta_ec\n');
fprintf('-------------------------------------------------------------------\n');
fprintf('TRUE VALUES        %12.4f    %12.4f    %12.4f    %12.4f\n', trueparam([1,6,7,8]));
fprintf('-------------------------------------------------------------------\n');
fprintf('\n');
fprintf('   MEAN 2step-True %12.4f    %12.4f    %12.4f    %12.4f\n', mean_bmattrue([1,6,7,8]));
fprintf('\n');
fprintf(' MEDIAN 2step-True %12.4f    %12.4f    %12.4f    %12.4f\n', median_bmattrue([1,6,7,8]));
fprintf('\n');
fprintf('   S.E. 2step-True %12.4f    %12.4f    %12.4f    %12.4f\n', se_bmattrue([1,6,7,8]));
```

```
fprintf('\n');
fprintf('------------------------------------------------------------------\n');
fprintf('\n');
fprintf('   MEAN 2step-Freq %12.4f    %12.4f    %12.4f    %12.4f\n', mean_bmatNP([1,6,7,8], 1));
fprintf('\n');
fprintf(' MEDIAN 2step-Freq %12.4f    %12.4f    %12.4f    %12.4f\n', median_bmatNP([1,6,7,8], 1));
fprintf('\n');
fprintf('   S.E. 2step-Freq %12.4f    %12.4f    %12.4f    %12.4f\n', se_bmatNP([1,6,7,8], 1));
fprintf('\n');
fprintf('------------------------------------------------------------------\n');
fprintf('\n');
fprintf('   MEAN NPL-Freq   %12.4f    %12.4f    %12.4f    %12.4f\n', mean_bmatNP([1,6,7,8], end));
fprintf('\n');
fprintf(' MEDIAN NPL-Freq   %12.4f    %12.4f    %12.4f    %12.4f\n', median_bmatNP([1,6,7,8], end));
fprintf('\n');
fprintf('   S.E. NPL-Freq   %12.4f    %12.4f    %12.4f    %12.4f\n', se_bmatNP([1,6,7,8], end));
fprintf('\n');
fprintf('------------------------------------------------------------------\n');
fprintf('\n');
fprintf('   MEAN 2step-Logit%12.4f    %12.4f    %12.4f    %12.4f\n', mean_bmatSP([1,6,7,8], 1));
fprintf('\n');
fprintf(' MEDIAN 2step-Logit%12.4f    %12.4f    %12.4f    %12.4f\n', median_bmatSP([1,6,7,8], 1));
fprintf('\n');
fprintf('   S.E. 2step-Logit%12.4f    %12.4f    %12.4f    %12.4f\n', se_bmatSP([1,6,7,8], 1));
fprintf('\n');
fprintf('------------------------------------------------------------------\n');
fprintf('\n');
fprintf('   MEAN NPL-Logit  %12.4f    %12.4f    %12.4f    %12.4f\n', mean_bmatSP([1,6,7,8], end));
fprintf('\n');
fprintf(' MEDIAN NPL-Logit  %12.4f    %12.4f    %12.4f    %12.4f\n', median_bmatSP([1,6,7,8], end));
fprintf('\n');
fprintf('   S.E. NPL-Logit  %12.4f    %12.4f    %12.4f    %12.4f\n', se_bmatSP([1,6,7,8], end));
fprintf('\n');
fprintf('------------------------------------------------------------------\n');
fprintf('\n');
fprintf('   MEAN 2step-Random   %8.4f    %12.4f    %12.4f    %12.4f\n', mean_bmatR([1,6,7,8], 1));
fprintf('\n');
fprintf(' MEDIAN 2step-Random   %8.4f    %12.4f    %12.4f    %12.4f\n', median_bmatR([1,6,7,8], 1));
fprintf('\n');
fprintf('   S.E. 2step-Random   %8.4f    %12.4f    %12.4f    %12.4f\n', se_bmatR([1,6,7,8], 1));
fprintf('\n');
fprintf('------------------------------------------------------------------\n');
fprintf('\n');
fprintf('   MEAN NPL-Random %12.4f    %12.4f    %12.4f    %12.4f\n', mean_bmatR([1,6,7,8], end));
fprintf('\n');
fprintf(' MEDIAN NPL-Random %12.4f    %12.4f    %12.4f    %12.4f\n', median_bmatR([1,6,7,8], end));
```

```
fprintf('\n');
fprintf('   S.E. NPL-Random %12.4f    %12.4f    %12.4f    %12.4f\n', se_bmatR([1,6,7,8], end));
fprintf('\n');
fprintf('-------------------------------------------------------------------------\n');

% TABLE 5: SQUARE-ROOT MEAN SQUARE ERRORS OF DIFFERENT ESTIMATORS
%          RATIOS OVER THE SQUARE-ROOT MSE OF THE 2-STEP PML USING THE TRUE CCPs

% Empirical Square-root MSE
trueparam = repmat(trueparam(1:kparam),1,npliter);
srmse_true = sqrt((mean_bmattrue'-trueparam(:,1)).^2 + se_bmattrue'.^2);
srmse_NP   = sqrt((mean_bmatNP-trueparam).^2 + se_bmatNP.^2);
srmse_SP   = sqrt((mean_bmatSP-trueparam).^2 + se_bmatSP.^2);
srmse_R    = sqrt((mean_bmatR-trueparam).^2 + se_bmatR.^2);

% Ratios
srmse_true = repmat(srmse_true, 1, npliter);
srmse_NP   = srmse_NP ./ srmse_true;
srmse_SP   = srmse_SP ./ srmse_true;
srmse_R    = srmse_R ./ srmse_true;

fprintf('\n');
fprintf('*************************************************************************************\n');
fprintf('   MONTE CARLO EXPERIMENT #%d\n', selexper);
fprintf('   SQUARE-ROOT MEAN SQUARE ERRORS\n');
fprintf('   RATIOS OVER THE SQUARE-ROOT MSE OF THE 2-STEP PML USING THE TRUE CCPs\n');
fprintf('\n');
fprintf('   TABLE 5 OF THE PAPER AGUIRREGABIRIA AND MIRA (2007)\n');
fprintf('*************************************************************************************\n');
fprintf('\n');
fprintf('-------------------------------------------------------------------------------------\n');
fprintf('                     theta_fc_1        theta_rs        theta_rn        theta_ec\n');
fprintf('-------------------------------------------------------------------------------------\n');
fprintf('SQ-MSE 2-step-TRUE %12.4f    %12.4f    %12.4f    %12.4f\n', srmse_true([1,6,7,8], 1));
fprintf('-------------------------------------------------------------------------------------\n');
fprintf('\n');
fprintf(' RATIO: 2step-Freq %12.4f    %12.4f    %12.4f    %12.4f\n', srmse_NP([1,6,7,8], 1));
fprintf('\n');
fprintf('-------------------------------------------------------------------------------------\n');
fprintf('\n');
fprintf('   RATIO: NPL-Freq %12.4f    %12.4f    %12.4f    %12.4f\n', srmse_NP([1,6,7,8], end));
fprintf('\n');
fprintf('-------------------------------------------------------------------------------------\n');
fprintf('\n');
fprintf('RATIO: 2step-Logit %12.4f    %12.4f    %12.4f    %12.4f\n', srmse_SP([1,6,7,8], 1));
fprintf('\n');
```

```
fprintf('-------------------------------------------------------------------------------\n');
fprintf('\n');
fprintf('  RATIO: NPL-Logit %12.4f    %12.4f    %12.4f    %12.4f\n', srmse_SP([1,6,7,8], end));
fprintf('\n');
fprintf('-------------------------------------------------------------------------------\n');
fprintf('\n');
fprintf('RATIO: 2step-Rando %12.4f    %12.4f    %12.4f    %12.4f\n', srmse_R([1,6,7,8], 1));
fprintf('\n');
fprintf('-------------------------------------------------------------------------------\n');
fprintf('\n');
fprintf(' RATIO: NPL-Random %12.4f    %12.4f    %12.4f    %12.4f\n', srmse_R([1,6,7,8], end));
fprintf('\n');
fprintf('-------------------------------------------------------------------------------\n');
```

Finally, we close the log file.

```
diary off;
```

Now that the main control script is complete, we now define several additional functions that carry out various steps above.

2. Computation of Markov Perfect Equilibrium (`mpeprob`)

The `mpeprob` function computes players' equilibrium conditional choice probabilities (CCPs) using the policy iteration algorithm.

Usage:

```
[ prob, psteady, mstate, dconv ] = mpeprob(inip, maxiter, param)
```

Inputs:

- `inip` - Matrix of initial choice probabilities with `numx` rows for each state and `nplayer` columns for each player.
- `maxiter` - Maximum number of policy iterations.
- `param` - Structure containing parameter values and settings.

Outputs:

- `prob` - Matrix of MPE entry probabilities with `numx` rows and `nplayer` columns. Each row corresponds to a state and each column corresponds to a player.
- `psteady` - Column vector with `numx` rows containing the steady-state distribution of (s_t, a_{t-1}).
- `mstate` - Matrix with `numx` rows and `nplayer+1` columns containing the values of state variables (s_t, a_{t-1}). The states in the rows are ordered as in the matrix `prob` above.
- `dconv` - Indicator for convergence: `dconv = 1` indicates that convergence was achieved and `dconv = 0` indicates no convergence.

As an example, when `sval = [ 1, 2 ]` and there are $N = 3$ players, the 16 rows of the `prob` matrix correspond to rows of the `mstate` vector as follows:

Row	s_t	$a_{1,t-1}$	$a_{2,t-1}$	$a_{3,t-1}$
1	1	0	0	0
2	1	0	0	1
3	1	0	1	0
4	1	0	1	1
5	1	1	0	0
6	1	1	0	1
7	1	1	1	0
8	1	1	1	1
9	2	0	0	0
10	2	0	0	1
11	2	0	1	0
12	2	0	1	1
13	2	1	0	0
14	2	1	0	1
15	2	1	1	0
16	2	1	1	1

```
function [ prob1, psteady1, mstate, dconv ] = mpeprob(inip, maxiter, param)
```

The function begins by calculating and storing the dimensions of the problem based on the input values.

```
nums = size(param.sval, 1);
nplayer = size(inip, 2);
numa = 2^nplayer;
numx = nums * numa;
```

Then it prints an informative header.

```
if (param.verbose)
    fprintf('\n\n');
    fprintf('********************************************************************************\n');
    fprintf('   COMPUTING A MPE OF THE DYNAMIC GAME\n');
    fprintf('********************************************************************************\n');
    fprintf('\n');
    fprintf('--------------------------------------------------------------------------------\n');
    fprintf('       Values of the structural parameters\n');
    fprintf('\n');
    for i = 1:nplayer
        fprintf('                       Fixed cost firm %d   = %12.4f\n', i, param.theta_fc(i));
    end
    fprintf('       Parameter of market size (theta_rs) = %12.4f\n', param.theta_rs);
    fprintf('Parameter of competition effect (theta_rn) = %12.4f\n', param.theta_rn);
    fprintf('                     Entry cost (theta_ec) = %12.4f\n', param.theta_ec);
    fprintf('                       Discount factor     = %12.4f\n', param.disfact);
    fprintf('                    Std. Dev. epsilons     = %12.4f\n', param.sigmaeps);
    fprintf('\n');
    fprintf('--------------------------------------------------------------------------------\n');
    fprintf('\n');
    fprintf('       BEST RESPONSE MAPPING ITERATIONS\n');
    fprintf('\n');
end
```

2.1. Construct the `mstate` *Matrix with Values of* s_t, a_{t-1}

First, we build a matrix named `aval` where the columns are all possible a_{t-1} vectors. Then we augment the `aval` matrix with a vector of possible s_t values to construct the `mstate` matrix.

```
aval = zeros(numa, nplayer);
for i = 1:nplayer
  aval(:,i) = kron(kron(ones(2^(i-1),1), [ 0; 1 ]), ones(2^(nplayer - i), 1));
end
```

```
mstate = zeros(numx, nplayer + 1);
mstate(:, 1) = kron(param.sval, ones(numa, 1));
mstate(:, 2:nplayer+1) = kron(ones(nums, 1), aval);
```

The resulting `mstate` matrix has the form described above. For example, with two s_t states and two players:

```
mstate =

   1   0   0
   1   0   1
   1   1   0
   1   1   1
   2   0   0
   2   0   1
   2   1   0
   2   1   1
```

2.2. *Initializing the Vector of Probabilities*

Next, we initialize the `prob0` vector, for storing the equilibrium CCPs, to the initial `inip` matrix provided. We also define two tolerances for the two termination criteria: `critconv` is the tolerance for the change in the sup norm of the CCP matrix and `criter` is the maximum number of iterations.

```
prob0 = inip;
critconv = (1e-3)*(1/numx);
criter = 1000;
dconv = 1;
```

2.3. *Iterative algorithm*

The iterative algorithm keeps track of the number of iterations, `iter`, and terminates when either the change in the sup norm of the CCP matrix is smaller than `critconv` or the maximum number of iterations exceeds `criter`.

```
iter = 1;
while ((criter > critconv) && (iter <= maxiter));
  if (param.verbose)
      fprintf('         Best response mapping iteration  = %d\n', iter);
```

```
    fprintf('         Convergence criterion = %g\n', criter);
    fprintf('\n');
  end
  prob1 = prob0;
```

2.3.1. *Matrix of transition probs* $\Pr(a_t|s_t, a_{t-1})$

```
ptrana = (prob1(:,1).^(aval(:,1)')).*((1-prob1(:,1)).^(1-aval(:,1)'));
i = 2;
while i <= nplayer;
  ptrana = ptrana.*(prob1(:,i).^(aval(:,i)')).*((1-prob1(:,i)).^(1-aval(:,i)'));
  i = i + 1;
end
```

Then for each player i we carry out the following steps:

```
for i = 1:nplayer
```

2.3.2. *Matrices* $\Pr(a_t|s_t, a_{t-1}, a_{it})$

```
mi = aval(:,i)';
ppi = prob1(:,i);
iptran0 = (1-mi)./((ppi.^mi).*((1-ppi).^(1-mi)));
iptran0 = ptrana .* iptran0;
iptran1 = mi./((ppi.^mi).*((1-ppi).^(1-mi)));
iptran1 = ptrana .* iptran1;
clear mi;
```

2.3.3. *Computing* $h_i = E[ln(N_{-it} + 1)]$

```
hi = aval;
hi(:,i) = ones(numa, 1);
hi = iptran1 * log(sum(hi, 2));
```

2.3.4. *Matrices with Expected Profits of Firm i*

```
profit1 = param.theta_fc(i) ...
        + param.theta_rs * mstate(:,1) ...
```

```
        - param.theta_ec*(1-mstate(:,i+1)) ... % Use i+1 because first component is market state
        - param.theta_rn * hi;              % Profit if firm is active
profit0 = zeros(numx,1);              % Profit if firm is not active
```

2.3.5. Transition Probabilities for Firm i

$\Pr(x_{t+1}, a_t | x_t, a_{t-1}, a_{i,t} = 0)$, $\Pr(x_{t+1}, a_t | x_t, a_{t-1}, a_{i,t} = 1)$

```
iptran0 = (kron(param.ptrans, ones(numa,numa))) .* (kron(ones(1,nums), iptran0));
iptran1 = (kron(param.ptrans, ones(numa,numa))) .* (kron(ones(1,nums), iptran1));
```

2.3.6. Computing Value Function for Firm i

```
v0 = zeros(numx, 1);
cbell = 1000;
while (cbell > critconv);
  v1 = [ profit0 + param.disfact * iptran0 * v0, ...
         profit1 + param.disfact * iptran1 * v0 ];
  v1 = v1 ./ param.sigmaeps;
  maxv1 = max(v1, [], 2);
  v1 = v1 - kron([ 1, 1 ], maxv1);
  v1 = param.sigmaeps * ( maxv1 + log(exp(v1(:,1)) + exp(v1(:,2))) );
  cbell = max(abs(v1 - v0), [], 1);
  v0 = v1;
end
```

2.3.7. Updating Probabilities for Firm i

```
v1 = [ profit0 + param.disfact * iptran0 * v0, ...
       profit1 + param.disfact * iptran1 * v0 ];
v1 = v1 ./ param.sigmaeps;
maxv1 = max(v1, [], 2);
v1 = v1 - repmat(maxv1, 1, 2);
prob1(:,i) = exp(v1(:,2))./(exp(v1(:,1))+exp(v1(:,2)));
```

This is the last player-i-specific step, so we terminate the loop over i.

```
end
```

We then evaluate the two convergence criteria–`criter`, the sup norm of the difference in CCPs, and `iter`, the iteration counter. In preparing for the next loop, we copy the updated CCPs in `prob1` over the old CCP matrix `prob0`.

```
criter = max(max(abs(prob1 - prob0)));
prob0 = prob1;
iter = iter + 1;
```

Finally, we terminate the **`while`** loop started in step 2.3.

```
end
clear iptran0 iptran1 v0 v1;
```

2.4. Reporting

Before returning, we print some informative output.

```
if (criter > critconv)
  dconv = 0;
  psteady1 = 0;
  if (param.verbose)
      fprintf('-------------------------------------------------------------------------------\n');
      fprintf('         CONVERGENCE NOT ACHIEVED AFTER %d BEST RESPONSE ITERATIONS\n', iter);
      fprintf('-------------------------------------------------------------------------------\n');
  end
else
  ptrana = ones(numx, numa);
  for i = 1:nplayer;
    mi = aval(:,i)';
    ppi = prob1(:,i);
    ppi1 = repmat(ppi, 1, numa);
    ppi0 = 1 - ppi1;
    mi1 = repmat(mi, numx, 1);
    mi0 = 1 - mi1;
    ptrana = ptrana .* (ppi1 .^ mi1) .* (ppi0 .^ mi0);
  end
  ptrana = (kron(param.ptrans, ones(numa, numa))) .* (kron(ones(1,nums), ptrana));
  criter = 1000;
  psteady0 = (1/numx) * ones(numx, 1);
  while (criter > critconv)
    psteady1 = ptrana' * psteady0;
```

```
    criter = max(abs(psteady1 - psteady0), [], 1);
    psteady0 = psteady1;
  end
  if (param.verbose)
      fprintf('-----------------------------------------------------------------------------\n');
      fprintf('         CONVERGENCE ACHIEVED AFTER %d BEST RESPONSE ITERATIONS\n', iter);
      fprintf('-----------------------------------------------------------------------------\n');
      fprintf('         EQUILIBRIUM PROBABILITIES\n');
      prob1
      fprintf('-----------------------------------------------------------------------------\n');
      fprintf('         STEADY STATE DISTRIBUTION\n');
      psteady1
      fprintf('-----------------------------------------------------------------------------\n');
  end
end
```

This completes the `mpeprob` function.

```
end
```

3. Procedure to Simulate Data from the Computed Equilibrium (`simdygam`)

The `simdygam` function simulates data of state and decision variables from the steady-state distribution of a Markov Perfect Equilibrium in a dynamic game of firms' market entry/exit with incomplete information.

Usage:

```
[ aobs, aobs_1, sobs, xobs ] = simdygam(nobs, pchoice, psteady, mstate)
```

Inputs:

- `nobs` - Number of simulations (markets)
- `pchoice` - Matrix of MPE probabilities of entry with `nstate` rows, for each state, and `nplayer` columns, for each player.
- `psteady` - Column vector with `nstate` rows containing the steady-state distribution of (s_t, a_{t-1}).

- `mstate` - Matrix with `nstate` rows and `nplayer + 1` columns containing the values of state variables (s_t, a_{t-1}). The states in the rows are ordered as in the `mpeprob` function defined above.

Outputs:

- `aobs` - Matrix with `nobs` rows and `nplayer` columns containing players' observed choices.
- `aobs_1` - Matrix with `nobs` rows and `nplayer` columns containing players' initial states.
- `sobs` - Column vector with `nobs` rows containing the simulated values of the market size state s_t.
- `xobs` - Column vector with `nobs` rows containing the indices of the resulting full state vectors. These are used as indices for the rows of the `mstate` matrix.

Examples of `aobs`, `aobs_1`, `xobs`, and `sobs` with `nobs = 6`:

```
aobs =

   1   1   1   1   1
   1   1   1   1   1
   0   1   0   0   0
   0   0   0   0   1
   0   1   1   1   1
   1   1   1   1   1

aobs_1 =

   1   1   1   0   1
   1   1   1   1   1
   1   1   1   1   1
   0   0   1   0   0
   0   1   0   1   1
   1   1   1   1   1

xobs =

   126
    96
    32
     5
```

```
    76
   160

sobs =

   4
   3
   1
   1
   3
   5
```

```
function [ aobs, aobs_1, sobs, xobs ] = simdygam(nobs, pchoice, psteady, mstate)
```

The function begins by calculating and storing the dimensions of the problem based on the input values.

```
nplay = size(pchoice, 2);
nums = size(pchoice, 1);
numa = 2^nplay;
numx = nums / numa;
```

3.1. Generating Draws from the Ergodic Distribution of (s_t, a_{t-1})

First, we construct the CDF in `pbuff1` by taking the cumulative sum of the steady state probabilities (`psteady`) across states. Then, we shift the CDF right by one state. We can then draw from the CDF by checking to see if a uniform draw falls in the interval defined by the CDF and shifted CDF.

```
pbuff1 = cumsum(psteady);
pbuff0 = cumsum([ 0; psteady(1:nums-1) ]);
uobs = rand(nobs, 1);
pbuff1 = kron(pbuff1, ones(1, nobs));
pbuff0 = kron(pbuff0, ones(1, nobs));
uobs = kron(uobs, ones(1, nums));
uobs = (uobs>=(pbuff0')) .* (uobs<=(pbuff1'));
```

Given the indicators for which intervals the uniform draws fall in, we draw indices `xobs` for the `mstate` matrix, which lists all the possible states of the model. Then we take

the first components of the states, the market sizes, and store them in `sobs`. Similarly, the last `nplay` observations are the firm activity indicators, stored as `aobs_1`.

```
xobs = uobs * [ 1:nums ]';
sobs = mstate(xobs, 1);
aobs_1 = mstate(xobs, 2:nplay+1);
clear pbuff0 pbuff1;
```

3.2. Generating Draws of a_t given (s_t, a_{t-1})

Now that we have simulated the state configurations, we calculate the choice probabilities and simulate new actions for each firm, returned in `aobs`.

```
pchoice = pchoice(xobs,:);
uobs = rand(nobs, nplay);
aobs = (uobs <= pchoice);
```

This completes the `simdygam` function.

```
end
```

4. Frequency Estimation of Choice Probabilities (`freqprob`)

This procedure obtains a frequency estimates of $\Pr(Y \mid X)$ where Y is a vector of binary variables and X is a vector of discrete variables.

Usage:

```
freqp = freqprob(yobs, xobs, xval)
```

Inputs:

- `yobs` - (`nobs` $\times$ `q`) vector with sample observations of $Y = [Y_1, Y_2, \ldots, Y_q]$.
- `xobs` - (`nobs` $\times$ `k`) matrix with sample observations of X.
- `xval` - (`numx` $\times$ `k`) matrix with the values of X for which we want to estimate $\Pr(Y \mid X)$.

Outputs:

- `freqp` - (`numx` $\times$ `q`) vector with frequency estimates of $\Pr(Y \mid X)$ for each value in `xval`: $\big(\Pr(Y_1 = 1 \mid X), \Pr(Y_2 = 1 \mid X), \ldots, \Pr(Y_q = 1 \mid X)\big)$.

```
function prob1 = freqprob(yobs, xobs, xval)
  numx = size(xval, 1);
  numq = size(yobs, 2);
  numobs = size(xobs, 1);
  prob1 = zeros(numx, numq);
  for t = 1:numx
    xvalt = kron(ones(numobs,1), xval(t,:));
    selx = prod(xobs == xvalt, 2);
    denom = sum(selx);
    if (denom == 0)
      prob1(t,:) = zeros(1, numq);
    else
      numer = sum(kron(selx, ones(1,numq)) .* yobs);
      prob1(t,:) = (numer') ./ denom;
    end
  end
end
```

5. Estimation of a Logit Model by Maximum Likelihood

These two procedures estimate a logit model by maximum likelihood using Newton's method for optimization.

5.1. Loglikelihood function for a logit model

This function calculates a loglikelihood function given a matrix of binary dependant variable, Y, a matrix of discrete independent variables, X, and values of parameters, θ.

Usage:

```
llik = loglogit(ydum, x, b)
```

Inputs:

- `ydum` - (`nobs` $\times$ `q`) vector of observations of the dependent variable.
- `x` - (`nobs` $\times$ `k`) matrix of explanatory variables.

Outputs:

- `llik` - Scalar with value of loglikelihood function.

```
function llik = loglogit(ydum, x, b)
  myzero = 1e-12;
  expxb = exp(-x*b);
  Fxb = 1./(1 + expxb);
  Fxb = Fxb + (myzero - Fxb).*(Fxb < myzero) ...
            + (1-myzero - Fxb).*(Fxb > 1 - myzero);
  llik = ydum'*ln(Fxb) + (1 - ydum)'*ln(1 - Fxb);
end
```

5.2. Estimation of a logit model by maximum likelihood (`milogit`)

This function obtains the maximum likelihood estimates of a binary logit model using Newton's method as the optimization algorithm.

Usage:

```
[ best, varest ] = milogit(ydum, x)
```

Inputs:

- `yobs` - (`nobs` $\times$ `q`) vector with sample observations of $Y = [Y_1, Y_2, \ldots, Y_q]$.
- `xobs` - (`nobs` $\times$ `k`) matrix with sample observations of X.
- `xval` - (`numx` $\times$ `k`) matrix with the values of X for which we want to estimate $\Pr(Y \mid X)$.

Outputs:

- `best` - (`k` $\times$ 1) vector with maximum likelihood estimates
- `var` - (`k` $\times$ `k`) vector with estimated variances-covariances of estimates

```
function [ b0, Avarb ] = milogit(ydum, x)
```

First, we set the constants for tolerance level and convergence criteria. Then, we define the starting values for parameters to be a (k $\times$ 1) vector of zeros. `lsopts.SYM` and `lsopts.POSDEF` is used to exploit the fact that the Hessian is a symmetric positive definite matrix, which can increase the speed.

```
nobs = size(ydum, 1);
nparam = size(x, 2);
eps1 = 1e-4;
eps2 = 1e-2;
b0 = zeros(nparam, 1);
iter = 1;
criter1 = 1000;
criter2 = 1000;
lsopts.SYM = true; lsopts.POSDEF = true;
```

We obtain the maximum likelihood estimates using Newton's method. We evaluate the two convergence criteria –`criter1` for the norm of differences in estimates and `criter2` for the norm of gradient.

```
while ((criter1 > eps1) || (criter2 > eps2))
  % fprintf("\n");
  % fprintf("Iteration               = %d\n", iter);
  % fprintf("Log-Likelihood function  = %12.4f\n", loglogit(ydum,x,b0));
  % fprintf("Norm of b(k)-b(k-1)      = %12.4f\n", criter1);
  % fprintf("Norm of Gradient         = %12.4f\n", criter2);
  % fprintf("\n");
  expxb0 = exp(-x*b0);
  Fxb0 = 1./(1+expxb0);
  dlogLb0 = x'*(ydum - Fxb0);
  d2logLb0 = ( repmat(Fxb0 .* (1-Fxb0), 1, nparam) .* x )'*x;
  b1 = b0 + linsolve(d2logLb0, dlogLb0, lsopts);
  criter1 = sqrt( (b1-b0)'*(b1-b0) );
  criter2 = sqrt( dlogLb0'*dlogLb0 );
  b0 = b1;
  iter = iter + 1;
end

expxb0 = exp(-x*b0);
Fxb0 = 1./(1+expxb0);
Avarb = -d2logLb0;
Avarb = inv(-Avarb);
```

It is possible to obtain the value of the log-likelihood function at the maximum likelihood estimates using `loglogit`.

```
  % sdb = sqrt(diag(Avarb));
  % tstat = b0./sdb;
  % llike = loglogit(ydum,x,b0);
  % numy1 = sum(ydum);
  % numy0 = nobs - numy1;
  % logL0 = numy1*log(numy1) + numy0*log(numy0) - nobs*log(nobs);
  % LRI = 1 - llike/logL0;
  % pseudoR2 = 1 - ( (ydum - Fxb0)'*(ydum - Fxb0) )/numy1;
end
```

6. Maximum Likelihood estimation of McFadden's Conditional Logit (`clogit`)

This procedure maximizes the pseudo likelihood function using Newton's method with analytical gradient and hessian.

Usage:

```
[best, varest] = clogit(ydum, x, restx)
```

Inputs:

- `ydum` - (`nobs` $\times$ 1) vector of observations of dependent variable which is a categorical variable with values: $\{1, 2, ..., nalt\}$
- `x` - (`nobs` $\times$ (`k` * `nalt`)) matrix of explanatory variables associated with unrestricted parameters. First k columns correspond to alternative 1, and so on.
- `restx` - (`nobs` $\times$ `nalt`) vector of the sum of the explanatory variables whose parameters are restricted to be equal to 1.

Outputs:

- `best` - (`k` $\times$ 1) vector with ML estimates
- `varest` - (`k` $\times$ `k`) matrix with estimate of covariance matrix.

```
function [ b0, Avarb ] = clogit(ydum, x, restx)
```

We first set the convergence criteria for the norm of differences in parameter estiamtes and define the size of observations and parameters.

```
cconvb = 1e-6;
myzero = 1e-16;
nobs = size(ydum, 1);
nalt = max(ydum);
npar = size(x, 2) / nalt;
lsopts.SYM = true;
lsopts.POSDEF = true;
```

This part calculates the sum of product of Y and X needed for analytical gradient.

```
xysum = 0;
for j = 1:nalt
  xysum = xysum + sum( repmat(ydum==j, 1, npar) .* x(:,npar*(j-1)+1:npar*j) );
end
```

We obtain the maximum likelihood estimates using Newton's method providing the analytical gradient and Hessian.

```
iter = 1;
criter = 1000;
llike = -nobs;
b0 = zeros(npar, 1);
while (criter > cconvb)
  % fprintf('\n');
  % fprintf('Iteration                = %d\n', iter);
  % fprintf('Log-Likelihood function  = %12.4f\n', llike);
  % fprintf('Norm of b(k)-b(k-1)      = %12.4f\n', criter);
  % fprintf('\n');

  % Computing probabilities
  phat = zeros(nobs, nalt);
  for j = 1:nalt
    phat(:,j) = x(:,npar*(j-1)+1:npar*j)*b0 + restx(:,j);
  end
  phatmax = repmat(max(phat, [], 2), 1, nalt);
  phat = phat - phatmax;
  phat = exp(phat) ./ repmat(sum(exp(phat), 2), 1, nalt);

  % Computing xmean
```

```
    sumpx = zeros(nobs, npar);
    xxm = 0;
    llike = 0;
    for j = 1:nalt
      xbuff = x(:,npar*(j-1)+1:npar*j);
      sumpx = sumpx + repmat(phat(:,j), 1, npar) .* xbuff;
      xxm = xxm + (repmat(phat(:,j), 1, npar).*xbuff)'*xbuff;
      llike = llike ...
              + sum( (ydum==j) ...
                     .* log( (phat(:,j) > myzero).*phat(:,j) ...
                             + (phat(:,j) <= myzero).*myzero) );
    end

    % Computing gradient
    d1llike = xysum - sum(sumpx);

    % Computing hessian
    d2llike = - (xxm - sumpx'*sumpx);

    % Gauss iteration
    b1 = b0 + linsolve(-d2llike, d1llike', lsopts);
    criter = sqrt( (b1-b0)'*(b1-b0) );
    b0 = b1;
    iter = iter + 1;
  end

  Avarb  = inv(-d2llike);
  % sdb    = sqrt(diag(Avarb));
  % tstat  = b0./sdb;
  % numyj  = sum(kron(ones(1,nalt), ydum)==kron(ones(nobs,1),(1:nalt)));
  % logL0  = sum(numyj.*log(numyj./nobs));
  % lrindex = 1 - llike/logL0;
end
```

7. NPL Algorithm

The `npldygam` implements the procedure that estimates the structural parameters of dynamic game of firms' entry/exit using the Nested Pseudo-Likelihood (NPL) algorithm.

Usage:

```
[best, varb] = npldygam(aobs, zobs, aobs_1, zval, ptranz, pchoice, bdisc, kiter)
```

Inputs:

- `aobs` - (`nobs` $\times$ `nplayer`) matrix with observations of firms' activity decisions (1 = active; 0 = inactive).
- `zobs` - (`nobs` $\times$ 1) vector with observations of market exogenous characteristics.
- `aobs_1` - (`nobs` $\times$ `nplayer`) matrix with observations of firms' initial states (1 = incumbent; 0 = potential entrant).
- `zval` - (`numz` $\times$ 1) vector with values of market characteristics.
- `ptranz` - (`numz` $\times$ `numz`) matrix with observations of firms' initial states (1 = incumbent; 0 = potential entrant).
- `pchoice` - ((`numz*2^nplayer`) $\times$ `nplayer`) matrix of players' choice probabilities used to initialize the procedure.
- `ptranz` - (`numz` $\times$ `numz`) matrix with observations of firms' initial states (1 = incumbent; 0 = potential entrant).
- `bdisc` - Discount factor.
- `kiter` - Number of NPL iterations

Outputs:

- `best` - (`k` $\times$ `kiter`) matrix with estimates of parameters $(\theta_{FC,1}, \theta_{FC,2}, \ldots, \theta_{FC,5}, \theta_{RS}, \theta_{RN}, \theta_{EC})$ for each iteration. The first column is the 1st-stage NPL, second column is the 2nd-stage NPL, and so on.
- `varest` - (`k` $\times$ `k`) matrix with estimated covariance matrices of the NPL estimators.

```
function [ best, varb ] = npldygam(aobs, zobs, aobs_1, zval, ptranz, pchoice, bdisc, kiter)
```

We start by setting constants including the number of observations (`nobs`), players (`nplayer`), possible firms' initial states (`numa`), possible market sizes (`numz`), states (`numx`), parameters to estimates (`kparam`). `best` and `varb` saves estimates and variances for each stage.

```
  eulerc = 0.5772;
  myzero = 1e-16;
  nobs = size(aobs, 1);
  nplayer = size(aobs, 2);
```

```
numa = 2^nplayer;
numz = size(zval, 1);
numx = numz*numa;
kparam = nplayer + 3;
best = zeros(kparam, kiter);
varb = zeros(kparam, kparam*kiter);
```

7.1. Construct the `mstate` *Matrix with Values of* s_t, a_{t-1}.

We construct the `mstate` matrix with state vectors as in `mpeprob` function. As described above, `mstate` is a (`numx` x `nplayer + 1`) matrix where `numx` is the total number of states in the model and there are columns for the market size state and each player's state.

```
aval = zeros(numa, nplayer);
for i = 1:nplayer
  aval(:,i) = kron(ones(2^(i-1),1), kron([ 0; 1 ], ones(2^(nplayer-i),1)));
end
mstate = zeros(numx, nplayer+1);
mstate(:,1) = kron(zval, ones(numa, 1));
mstate(:,2:nplayer+1) = kron(ones(numz, 1), aval);
```

7.2. Assign each state a state index (`indobs`*)*

We label each observation with the corresponding state index by constructing `indobs`, a vector of length `nobs`. Each entry in the vector is an integer from `1` to `numx`, and corresponds to a row of the `mstate` matrix. For example, when `nobs = 10` in the standard 5 player model, where `numx` is `160`:

```
indobs =

   158
   128
   126
   160
     2
    92
   160
   160
   128
    86
```

```
indzobs = (repmat(zobs, 1, numz)==repmat(zval', nobs, 1))*[ 1:numz ]';
twop = kron(ones(nobs, 1), 2.^(nplayer-[1:nplayer]));
indobs = sum(aobs_1.*twop, 2);
indobs = (indzobs-1).*(2^nplayer) + indobs + 1;
```

7.3. NPL algorithm

Our goal is to express the pseudo likelihood function in terms of parameters, θ. We start the NPL algorithm from defining matrices. `aobs` is a ((`nobs` $\times$ `nplayer`) $\times$ 1) matrix with entries equal to 2 for active firms and 1 for inactive firms. This is to match the requirements of `clogit` below. `u0` and `u1` are ((`numx` $\times$ `nplayer`) $\times$ `k`) matrices that store explanatory variables.

```
aobs = 1 + reshape(aobs, [nobs * nplayer, 1]);
u0 = zeros(numx * nplayer, kparam);
u1 = zeros(numx * nplayer, kparam);
e0 = zeros(numx * nplayer, 1);
e1 = zeros(numx * nplayer, 1);
```

We iterate the algorithm `kiter` times with a counter `iter`. The iteration output was been commented out for brevity.

```
for iter = 1:kiter
  % fprintf('\n');
  % fprintf(' -----------------------------------------------------\n');
  % fprintf(' POLICY ITERATION ESTIMATOR: STAGE = %d\n', iter);
  % fprintf(' -----------------------------------------------------\n');
  % fprintf('\n');
```

7.3.1. Matrix of transition probabilities $\Pr(a_t|s_t, a_{t-1})$

```
  ptrana = ones(numx, numa);
  for i = 1:nplayer;
    mi = aval(:,i)';
    ppi = pchoice(:,i);
    ppi1 = repmat(ppi, 1, numa);
    ppi0 = 1 - ppi1;
    mi1 = repmat(mi, numx, 1);
```

```
    mi0 = 1 - mi1;
    ptrana = ptrana .* (ppi1 .^ mi1) .* (ppi0 .^ mi0);
  end
```

7.3.2. *Storing* $(I - \beta F_x^P)$

We compute the transition probability of states from the point of view of firm i who knows her own action a_{it}, and the current market size s_t but does not know other firms' actions, denoted $f_i^P(x_{t+1}|s_t, x_t)$, and construct a matrix, F_x^P. Then, we pre-calculate $(I - \beta F_x^P)$ before estimation as it does not rely on θ.

```
i_bf = kron(ptranz, ones(numa, numa)) .* kron(ones(1,numz), ptrana);
i_bf = eye(numx) - bdisc * i_bf;
```

7.3.3. *Construction of explanatory variables*

`uobs0` and `uobs1` will be matrices of constructed explanatory variables of each observation for being inactive and active, respectively for the conditional logit estimation. `eobs0` and `eobs1` will store discounted and expected sums of ε_{it} for each action and each observation.

```
uobs0 = zeros(nobs*nplayer,kparam);
uobs1 = zeros(nobs*nplayer,kparam);
eobs0 = zeros(nobs*nplayer,1);
eobs1 = zeros(nobs*nplayer,1);
```

Then, for each player i, we follow these steps:

```
for i = 1:nplayer
```

7.3.4. *Matrices* $\Pr(a_t|s_t, a_{t-1}, a_{it})$

```
  mi = aval(:,i)';
  ppi = pchoice(:,i);
  ppi= (ppi>=myzero).*(ppi<=(1-myzero)).*ppi ...
     + (ppi<myzero).*myzero ...
     + (ppi>(1-myzero)).*(1-myzero);
  ppi1 = repmat(ppi, 1, numa);
```

```
ppi0 = 1 - ppi1;
mi1 = repmat(mi, numx, 1);
mi0 = 1 - mi1;
ptrani = ((ppi1 .^ mi1) .* (ppi0 .^ mi0));
ptranai0 = ptrana .* (mi0 ./ ptrani);
ptranai1 = ptrana .* (mi1 ./ ptrani);
clear mi;
```

7.3.5. Computing $h_i = E[\ln(N_{-it} + 1)]$

```
hi = aval;
hi(:,i) = ones(numa, 1);
hi = ptranai1 * log(sum(hi, 2));
```

7.3.6. Creating $Z_i^P(0)$ (`umat0`) and $Z_i^P(1)$ (`umat1`)

Firm i's profit at time t can be written as:

$$\begin{aligned}
\pi_{it}(0) &= \varepsilon_{it}(0) \\
&= z_{it}(0)\theta_i + \varepsilon_{it}(0) \\
\pi_{it}(1) &= \theta_{FC,i} + \theta_{RS}\ln(S_t) - \theta_{RN}\ln\left(1 + \sum_{j\neq i} a_{jt}\right) - \theta_{EC}(1 - a_{i,t-1}) + \varepsilon_{it}(1) \\
&= z_{it}(1)\theta_i + \varepsilon_{it}(1)
\end{aligned}$$

where z_{it} is a (kx1) vector of zeros, and $z_{it}(1) \equiv \left\{1, \ln(S_t), -\ln\left(1 + \sum_{j\neq i} a_{jt}\right), -(1 - a_{i,t-1})\right\}$ and $\theta_i \equiv (\theta_{FC,i}, \theta_{RS}, \theta_{RN}, \theta_{EC})$.

`umat0` and `umat1` represent $z_{it}(0)$ and $z_{it}(1)$, respectively.

```
umat0 = zeros(numx, nplayer+3);
umat1 = eye(nplayer);
umat1 = umat1(i,:);
umat1 = [ repmat(umat1, numx, 1), mstate(:,1), (-hi), (mstate(:,i+1)-1) ];
clear hi;
```

7.3.7. Creating Z_i^P (`sumu`) and λ_i^P (`sume`)

Then, denoting $P_i(x_t)$ is the conditional choice probability of staying active that maximizes the expected value of firm i, the one-period expected profit of firm i will be

$$\begin{aligned}\pi_{it}^P(a_{it}) &= (1 - P_i(x_t))[z_{it}(0)\theta_i + \varepsilon_{it}(0)] + P_i(x_t)[z_{it}(1)\theta_i + \varepsilon_{it}(1)] \\ &= z_{it}^P\theta_i + e_{it}^P\end{aligned}$$

where z_{it}^P and e_{it}^P are the expected values of $z_{it}(a_{it})$ and $\varepsilon_{it}(a_{it})$, respectively.

Since ε_{it} follows the T1EV distribution, $E\varepsilon_{it}$ is

$$e_{it}^P = \text{Euler's constant} - (1 - P_i(x_t))\ln(1 - P_i(x_t)) - P_i(x_t)\ln(P_i(x_t)).$$

```
ppi1 = kron(ppi, ones(1, nplayer+3));
ppi0 = 1 - ppi1;
sumu = ppi0.*umat0 + ppi1.*umat1;
sume = (1-ppi).*(eulerc-log(1-ppi)) + ppi.*(eulerc-log(ppi));
clear ppi ppi0 ppi1;
```

7.3.8. Creating `ww`

In Aguirregabiria and Mira (2007), a MPE is defined as a vector of choice probabilities where for every firm i,

$$P_i(x_t) = G_i([\tilde{z}_{it}^P(1) - \tilde{z}_{it}^P(0)]\theta_i - [\tilde{e}_{it}^P(1) - \tilde{e}_{it}^P(0)])$$

where $\tilde{z}_{it}^P$ is the expected and discounted sum of current and future z vectors, $\tilde{e}_{it}^P$ is the expected and discounted sum of realizations of ε_{it}, and G_i is the distribution of ε_{it} which is assumed as T1EV here.

$$\begin{aligned}\tilde{z}_{it}^P(a_{it}) &\equiv z_{it}^P(a_{it}) + E\Big(\sum_{s=1}^{\infty}\beta^s z_{i,t+s}^P(a_{i,t+s})|x_t, a_{it}\Big) \\ \tilde{e}_{it}^P(a_{it}) &\equiv E\Big(\sum_{s=1}^{\infty}\beta^s e_{i,t+s}^P(a_{i,t+s})|x_t, a_{it}\Big)\end{aligned}$$

Now we introduce $W_{z_i}^P$, a vector of expected $\tilde{z}_{it}^P(a_{it})$, and $W_{e_i}^P$, a scalar value for expected $\tilde{e}_{it}(a_{it})$.

$$\begin{aligned}W_{z_i}^P(x_t) &\equiv (1 - P_i(x_t))\tilde{z}_{it}^P(0) + P_i(x_t)\tilde{z}_{it}^P(1) \\ W_{e_i}^P(x_t) &\equiv (1 - P_i(x_t))\tilde{e}_{it}^P(0) + P_i(x_t)\tilde{e}_{it}^P(1)\end{aligned}$$

Using new notations, we can rewrite $\tilde{z}_{it}^P(a_{it})$ and $\tilde{e}_{it}^P(a_{it})$:

$$\tilde{z}_{it}^P(a_{it}) \equiv z_{it}^P(a_{it}) + \beta \sum_{x_{t+1} \in \mathcal{X}} f_i^P(x_{t+1}|x_t, a_{it}) W_{z_i}^P(x_{t+1})$$
$$\tilde{e}_{it}^P(a_{it}) \equiv \beta \sum_{x_{t+1} \in \mathcal{X}} f_i^P(x_{t+1}|x_t, a_{it}) W_{e_i}^P(x_{t+1}).$$

We can obtain the closed-form expression for $W_{z_i}^P$ and $W_{e_i}^P$, where they are a (`numx` $\times$ `k`) matrix with rows of $W_{z_i}^P(x_{t+1})$ and a (`numx` $\times$ `1`) matrix with rows of $W_{e_i}^P(x_{t+1})$:

$$W_{z_i}^P = (I - \beta F_X^P)^{-1}[(1 - P_i)Z_i^P(0) + P_i Z_i^P(1)]$$
$$W_{e_i}^P = (I - \beta F_X^P)^{-1} e_i^P.$$

In the code, we stack two equations and solve for $W_{z_i}^P$ and $W_{e_i}^P$ simultaneously. `ww` is a (`numk` $\times$ (`k`+`1`)) matrix of $W_{z_i}^P$ and $W_{e_i}^P$ stacked together.

```
ww = linsolve(i_bf, [ sumu, sume ]);
clear sumu sume;
```

7.3.9. Creating utilda and etilda

We can express $\tilde{z}_{it}^P(0)$ (`utilda0`), $\tilde{z}_{it}^P(1)$ (`utilda1`), $\tilde{e}_{it}^P(0)$ (`etilda0`), and $\tilde{e}_{it}^P(1)$ (`etilda1`) using $W_{z_i}^P$ and $W_{e_i}^P$ (`ww`).

```
ptranai0 = kron(ptranz, ones(numa, numa)) .* kron(ones(1,numz), ptranai0);
utilda0 = umat0 + bdisc*(ptranai0*ww(:,1:kparam));
etilda0 = bdisc*(ptranai0*ww(:,kparam+1));
clear umat0 ptranai0;

ptranai1 = kron(ptranz, ones(numa, numa)) .* kron(ones(1,numz), ptranai1);
utilda1 = umat1 + bdisc*(ptranai1*ww(:,1:kparam));
etilda1 = bdisc*(ptranai1*ww(:,kparam+1));
clear umat1 ptranai1;
```

7.3.10. Creating observations uobs and eobs

We now pick the values of $\tilde{z}_{it}^P$ and $\tilde{e}_{it}^P$ that correspond to each observation using state index matrix `indobs` and construct `uobs0`, `uobs1`, `eobs0`, and `eobs1`.

```
uobs0((i-1)*nobs+1:i*nobs,:) = utilda0(indobs,:);
uobs1((i-1)*nobs+1:i*nobs,:) = utilda1(indobs,:);
```

```
    eobs0((i-1)*nobs+1:i*nobs,:) = etilda0(indobs,:);
    eobs1((i-1)*nobs+1:i*nobs,:) = etilda1(indobs,:);
    u0((i-1)*numx+1:i*numx,:) = utilda0;
    u1((i-1)*numx+1:i*numx,:) = utilda1;
    e0((i-1)*numx+1:i*numx,:) = etilda0;
    e1((i-1)*numx+1:i*numx,:) = etilda1;
    clear utilda0 utilda1 etilda0 etilda1;
```

Create $\tilde{z}_i^P$ and $\tilde{e}_i^P$ for each player i.

```
end
```

7.4. Pseudo Maximum Likelihood Estimation

We can express the pseudo likelihood function using $\tilde{z}_i^P$ and $\tilde{e}_i^P$ obtained above.

$$Q(\theta, P) = \sum_{i=1}^{N}\sum_{m=1}^{M}\sum_{t=1}^{T} a_{imt}\Big(G_i\big([\tilde{z}_{it}^P(1) - \tilde{z}_{it}^P(0)]\theta_i - [\tilde{e}_{it}^P(1) - \tilde{e}_{it}^P(0)]\big)\Big) + (1 - a_{imt})\Big(1 - G_i\big([\tilde{z}_{it}^P(1) - \tilde{z}_{it}^P(0)]\theta_i - [\tilde{e}_{it}^P(1) - \tilde{e}_{it}^P(0)]\big)\Big)$$

Now we can obtain maximum likelihood estimates using the `clogit` function. The `best` matrix stores estimates for each iteration and `varb` matrix stores variances of estimates.

```
[ tetaest, varest ] = clogit(aobs, [uobs0, uobs1], [eobs0, eobs1]);
best(:,iter) = tetaest;
varb(:,(iter-1)*kparam+1:iter*kparam) = varest;
```

7.5. Updating probabilities

Update the choice probabilities for each player i using the maximum likelihood estimates from step 7.4. We assume that ε_i follows T1EV, so the conditional choice probability for being active is exponential of choice specific value divided by sum of 1 and the nominator.

$$\hat{P}_i^K(x_t) = G_i\left(\left[\tilde{z}_{it}^{\hat{P}^{K-1}}(1) - \tilde{z}_{it}^{\hat{P}^{K-1}}(0)\right]\hat{\theta}^K - \left[\tilde{e}_{it}^{\hat{P}^{K-1}}(1) - \tilde{e}_{it}^{\hat{P}^{K-1}}(0)\right]\right)$$

where $\hat{P}^{K-1}$ is the conditional choice probabilities obtained from the previous stage, and $\hat{\theta}^K$ is the parameter estimates from the first step in K-th iteration.

```
    for i = 1:nplayer
      buff = (u1((i-1)*numx+1:i*numx,:)-u0((i-1)*numx+1:i*numx,:));
      buff = buff*tetaest ...
           + (e1((i-1)*numx+1:i*numx,:)-e0((i-1)*numx+1:i*numx,:));
      pchoice(:,i) = exp(buff)./(1+exp(buff));
    end
```

We iterate the NPL algorithm for `npliter` times.

```
  end
```

This completes the NPL algorithm.

```
end
```

References

Aguirregabiria, V. (2009). Estimation of dynamic discrete games using the nested pseudo likelihood algorithm: Code and application. MPRA Paper 17329, University Library of Munich, University of Toronto. [1]

Aguirregabiria, V. and P. Mira (2007). Sequential estimation of dynamic discrete games. *Econometrica 75*, 1–53. [1, 8, 11, 37]

Blevins, J. R. and M. Kim (2019). Nested pseudo likelihood estimation of continuous-time dynamic discrete games. Working paper, The Ohio State University. [1]