# Problem Set 4: Estimation of dynamic discrete choice models

Due: November 29

Consider the following dynamic model of inventory control. The per-period payoff function is given by:

$$U(a|i,c,p,\epsilon) = \begin{cases} \alpha c - p + \epsilon_1 & \text{If } a = 1 \\ \alpha c + \epsilon_0 & \text{If } a = 0 \text{ and } i > 0 \\ \lambda 1(c > 0) + \epsilon_0 & \text{If } a = 0 \text{ and } i = 0 \end{cases}$$

where $c = \{0, 1\}$ is consumption shock, $i$ is the inventory level, and $\epsilon_j \sim T1EV(0, 1)$. The parameter $\lambda$ measures the *stockout* penalty: consumption shock is positive but inventory is zero.

The value function is given by:

$$\begin{aligned} V(i,c,p,\epsilon_t) &= \max_{a\in\{0,1\}} U(a|i,c,p,\epsilon) + \beta \sum_{c',p'} E_{\epsilon'}[V(i',c',p',\epsilon')] \Pr(c',p'|c,p,a) \qquad (1) \\ \text{s.t.} \quad & i' = \min\{\bar{i}, i + a - c\} \\ & c' = \begin{cases} 0 & \text{With probability } 1/2 \\ 1 & \text{With probability } 1/2 \end{cases} \\ & p' = \begin{cases} p_s & \text{With probability } \pi(p) \\ p_r & \text{With probability } 1 - \pi(p) \end{cases} \end{aligned}$$

The consumption and price state variables follow discrete Markov processes. The process for price is given by the following matrix:

$$\Pi = \begin{pmatrix} 1 - \pi(p_r) & \pi(p_r) \\ 1 - \pi(p_s) & \pi(p_s) \end{pmatrix} = \begin{pmatrix} 0.9 & 0.1 \\ 0.9 & 0.1 \end{pmatrix} \qquad (2)$$

where the first column corresponds to transitions to the regular price $p_r$, and the second column corresponds to transitions to the sales price $p_s$. The parameters are given by:

| $\lambda$ | $\alpha$ | $\beta$ | $p_r$ | $p_s$ | $\bar{i}$ |
|---|---|---|---|---|---|
| -4 | 2 | 0.99 | 4 | 1 | 8 |

The state-space includes $S = 36$ discrete points. Let $s_k$ denotes the state vector for point $k$. The following files contain the state-space and transition matrix:

- $S = \{s_1, \ldots, s_{36}\}$: PS4_state_space.csv
- $F(s'|s, a = 0)$: PS4_transition_a0.csv

- $F(s'|s, a = 1)$: PS4_transition_a1.csv
- Simulated data: PS4_simdata.csv

1. Write down the equation that defines the expected value function, $\bar{V}(s) = E_\epsilon[V(s, \epsilon)]$. Numerically solve for the expected value function using the value-function iteration algorithm, and tabulates its value against the discrete state variable $s$.
2. Use the simulated sequence of choices and states to calculate an estimate of the conditional-choice probability (CCP) vector $\hat{P}(s)$. Use a frequency estimator, and constraint the estimated frequency to be between 0.001 and 0.999. At the true parameters, calculate the implied expected value function $\bar{V}^{\hat{P}}(s)$ using the CCP mapping. Compare and tabulate this estimate with the true value function calculated in question 1. Discuss the difference(s).
3. Write the equation for the log-likelihood function.
4. Calculate the maximum-likelihood estimate of the parameter $\lambda$. Use the Nested-Fixed Point algorithm.

## Sample Code

```
/*****************************************/
/* Global variables */
decl alpha=2;
decl lambda=-3;
decl beta=0.99;
decl ibar=4;
decl ps=1/2;
decl pr=2;
decl c=1/4;
decl gamma=1/2;
decl mF1,mF0,mS;
/* Simulated choices and states */
decl vPhat; /* Estimated choice-probabilities Sx1 */
decl vY; /* Observed choices Nx1 */
decl vSid; /* Observed state IDs Nx1 */
/* State space indices */
enum{iI,iC,iP}; /* Columns identifiers for each state variable */
/* Parameter names */
enum{ialpha,ilambda}; /* Row identifiers for parameter */

/*****************************************/
/* Choice-specific value-function */
value(amV,const vEV)
{
  decl vU1=alpha*mS[][iC]-mS[][iP];
  decl vU0=alpha*mS[][iC].*(mS[][iI].>0)+lambda*(mS[][iC].>0).*(mS[][iI].==0);
  decl mV=(vU0+beta*mF0*vEV)~(vU1+beta*mF1*vEV);
  amV[0]=mV;
  return 1;
}
/* Expected value function */
emax(aEV,const vEV0)
{
  decl mV;
  value(&mV,vEV0);
  aEV[0]=log(sumr(exp(mV)))+M_EULER;
  return 1;
}
/* CCP mapping */
ccp(aEV,aP,const vP)
{
  decl vU1=alpha*mS[][iC]-mS[][iP];
  decl vU0=alpha*mS[][iC].*(mS[][iI].>0)+lambda*(mS[][iC].>0).*(mS[][iI].==0);
  decl vE1=M_EULER-log(vP);
```

```
  decl vE0=M_EULER-log(1-vP);
  decl mF=mF0.*(1-vP)+mF1.*vP;
  decl vEU=(1-vP).*(vU0+vE0)+vP.*(vU1+vE1);
  decl vEVp=invert(unit(rows(vP))-beta*mF)*vEU;
  decl mV;
  value(&mV,vEVp);
  aP[0]=exp(mV[][1])./sumr(exp(mV));
  aEV[0]=vEVp;
  return 1;
}
/* Likelihood function */
lfunc_ccp(const vP, const adFunc, const avScore, const amHessian)
{
  alpha=vP[ialpha];
  lambda=vP[ilambda];
  decl vCCP,vEV;
  ccp(&vEV,&vCCP,vPhat);
  vCCP=vCCP[vSid];
  decl vL=vCCP.*vY+(1-vCCP).*(1-vY);
  decl LLF;
  LLF=double(sumc(log(vL))/1000);
  adFunc[0]=LLF;
  return 1;
}
lfunc_nfxp(const vP, const adFunc, const avScore, const amHessian)
{
  alpha=vP[ialpha];
  lambda=vP[ilambda];
  decl vCCP,vCCP0;
  decl it=0;
  decl eps=10^(-10);
  vCCP=vPhat;
  do{
    vCCP0=vCCP;
    ccp(&vEV,&vCCP,vCCP0);
    it+=1;
  }while(norm(vCCP0-vCCP)>eps);
  vCCP=vCCP[vSid];
  decl vL=vCCP.*vY+(1-vCCP).*(1-vY);
  decl LLF;
  LLF=double(sumc(log(vL))/1000);
  adFunc[0]=LLF;
  return 1;
}
/*****************************************************/
```