Implementing Mixed Logit

Consider implementing a mixed logit model with:

- A single choice outcome per person
- A subset of the parameters distributed independent normal such that $\beta_n \sim N(\mu_\beta, \sigma_\beta)$.
- A subset of the parameters deterministic such that $\beta_n = \beta$
- Simulated random variables generated via Halton draws

Programming tasks:

- Construct an $m$-function for the simulated likelihood that simulates all relevant probabilities and returns the (negative) simulated log-likelihood value for current values of the parameters being estimated
- Construct a main program setting up the problem and calling the optimizer

**FUNCTION MXL\_LIKE.M**

Accepts as inputs:

- Data $Y$ and $X$ as usual
- Index identifying which utility function parameters are fixed
- Index identifying which utility function parameters are normal
- Matrix of random variables for use in simulation (generated from Halton sequences)
- Starting values for parameters to be estimated

Returns:

- (negative) of the simulated sample log-likelihood

**MAIN PROGRAM**

- Sets up data $X$ and $Y$ for particular problem
- Constructs Halton sequences for each normally distributed coefficient and transforms sequence to construct normal realizations
- Calls optimization routine, passing to `mxl_like` all needed inputs.

Example:

- $J=5$ and $k=4$ (i.e. we have four variables in the model)
- We assume $\beta_1$ and $\beta_3$ are fixed
- We assume $\beta_2$ and $\beta_4$ are normally distributed means and standard deviations $b_2$, $b_4$ and $s_2$, $s_4$ respectively.
function ll = mxl_like(b,Y,X,FC,NC,errors);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% A function to simulate the likelihood function for a specific version of
% the mixed logit model that contains fixed and independent random
% coefficients.
% Inputs include:
% b: sx1 coefficient vector
% Y: nxj matrix of observed outcomes
% X: nx(j*k) matrix of explanatory variables
% FC: NFCx1 vector indicating positions of fixed coefficients
% NC: NNCx1 vector indicating positions of normal coefficients
% errors: Rx(n*NNC) matrix of standard normal realizations
% The function returns the scalar simulated (negative) log-likelihood
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% establish dimensions for model
[R junk] = size(errors);
[n j] = size(Y);
[n jk] = size(X);
k = jk/j;
s = length(b);

NFC = length(FC);
NNC = length(NC);

% initialize indirect utility and probability matrices
p0 = zeros(n,j);
V = zeros(n,j);

% Add in fixed coefficients
% Add in normal coefficients and calculate simulated probabilities
% by looping over r=1,...R.

% Compute log-probabilities and construct (negative) log-likelihood
lnp = log(p0/R);
ll = -sum(sum((Y.*lnp)'));
load X; load Y;

% establish fixed and random coefficients
FC = [1; 3];
NC = [2; 4];
R = 100;

% Create Rx(N*d) error matrix, where d is the dimension of integration
d = 2;
prim = [2, 3, 5, 7, 11];

hm = [];
for c = 1:d
    h = halton(25+R*nobs,prim(c));
    h = h(26:end);
    h = reshape(h,R,nobs);
    hm = [hm, h];
end;
hm = norminv(hm,0,1);

% start values:  b1, b3, b2, s2, b4, s4
b0 = [.4; .5; 2.5; 1; 1.9; 1];

[b_ml,fval,exitflag,output,grad,hessian] = fminunc(@(d) mxl_like(d,Y,X,FC,NC,hm),b0);