Course: Econ 634, Fall 2017
Professor: Florian Kuhn.
Student: Luis D. Chancí A. (*lchanci1@binghamton.edu*)

Homework No 4. Aiyagari Model.

Part I. Definitions.

1. **Question 1. The Recursive Problem.**

Suggested Answer:

The firm's maximization problem is

$$\max_{\{K_{t+1}^d, L_t^d\}} \sum_{t=0}^{\infty} \left(\frac{1}{\Pi_{i=0}^t r_i} \right) \left[K_t^\alpha L_t^{1-\alpha} - w_t L_t - r_t K_t + (1-\delta) K_t \right]$$

The first order conditions,

$$\begin{aligned} r_t &= \alpha K_{t+1}^{\alpha-1} L_t^{1-\alpha} + (1-\delta) \\ w_t &= (1-\alpha) K_t^\alpha L_t^{-\alpha} \end{aligned}$$

And the recursive problem for consumers,

$$v(z,a) = \max_{a' \in \Psi(z,a)} \left\{ \frac{(zw\bar{l} + ra - a')^{1-\sigma}}{1-\sigma} + \beta * \mathbb{E}_{z'|z} \left[v(z', a') \right] \right\} \tag{1}$$

with,

$$\begin{aligned} \Psi(z,a) &= \left\{ a' : \underline{a} \leq a' \leq zw\bar{l} + ra \right\} \\ & \quad ln\, z_{t+1} = \rho\, ln\, z_t + \varepsilon_t \;\; ; \;\; \varepsilon_t \sim \mathbb{N}(0, \sigma^2) \end{aligned}$$

Part II. Computational.

In order to solve the model using the four different methods in Matlab (Search Grid, Policy Function Iteration, Linear Interpolation, and Cubic Spline Interpolation), and with the aim of having a well organized code, I created a number of scripts and functions, which I describe next and for which I have copied the Matlab code in the appendix section of this report.

- **Ch_U.m (Function):** This function calculates the matrix of consumptions and reports the utility function adjusted by a large negative value when consumption is negative.
- **Ch_VFI.m (Function):** This function does the value function iteration method.
- **Ch_Grid_Search.m (Script):** This script is an a loop for the grid search method.
- **Ch_PFI.m (Script):** This script does the policy function iteration.
- **Ch_Interpol_Linear.m (Script):** This script does the linear interpolation method.
- **Ch_Interpol_Spline.m (Function):** This script does the cubic spline interpolation method.
- **Ch_Mu_standard.m (Script):** This script is for the iteration over the distribution (as we did in the previous homework).
- **Ch_Mu_Spline.m (Script):** Considering that the optimization step, when using higher order polynomials for the interpolation function, may produce some values that are not part of the grid, I tried to adjust the policy function (e.g. let's say that for a number 5.7, we can say that 70% is 6 and 30% is 5). This script does the policy function adjustment and also the iteration over the distribution.
- **Ch_goldensectionsearch.m (Function):** I did minor modifications to the script Golden-SectionSearch.m. Specifically, I transformed the script into a function and I erased all the commands related to graphs.

In addition, I used the other two provided programs: Tauchen.m, and makeQmatrix.m

Results

- Aggregate Effective Labor Supply:

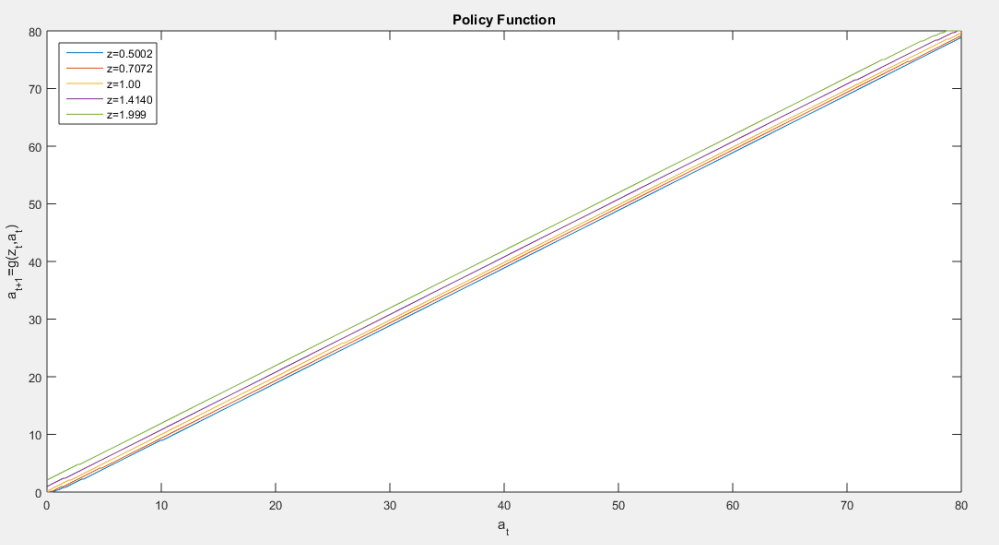
$$L^s = \int_z z\bar{l}\bar{\pi}(dz) \approx \underbrace{Z}_{(1x5)} * \underbrace{\bar{\pi}}_{(5x1)} = 1.0338$$

- For the discretization I used two different lengths. Although for both grids the minimum level is zero, $a_{min} = 0$, one grid is defined by using a size of $num\_a = 500$, and the second uses a shorter grid (in general, I used $num\_a0 = 10\% * num\_a$ but I tested $num\_a0 = 12$ for the cubic spline). The idea is to use num_a directly for the 'standard' methods, such as value function iteration and policy function iteration, and use num_a0 together with the VFI as initial guesses for the interpolation methods, and then extend the result to the larger grid (num_a).

lchanci1@binghamton.edu

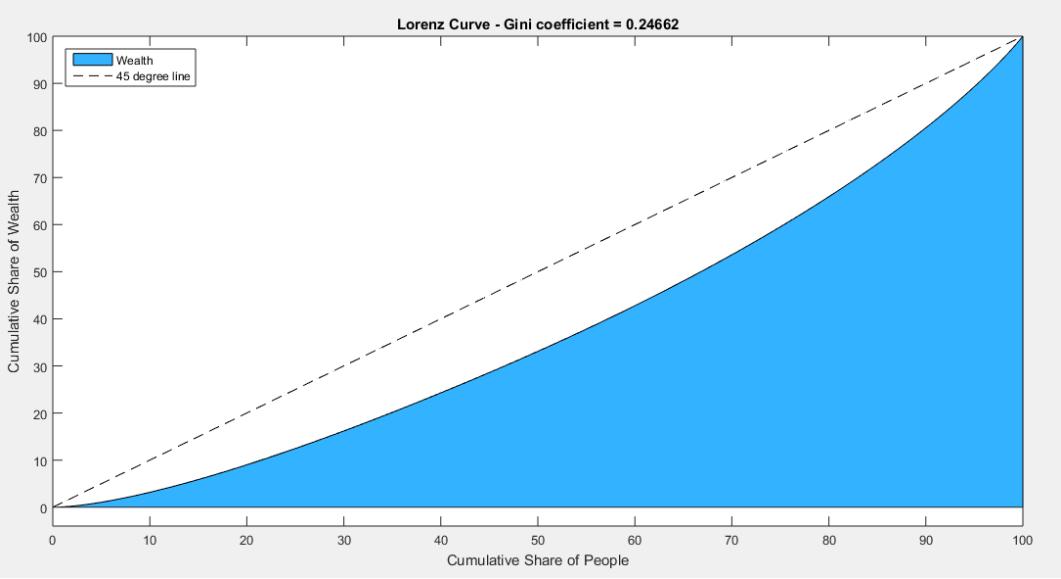
- Solution using value function iteration:
  - Aggregated Capital, $K = 30.4688$
- Analysis:
  - Steady state interest rate is 1.0099, which is lower to $r^{CM} = 1/\beta = 1.0101$.
  - Figure (1) shows the policy function for the $m = 5$ productivity states.

Figure 1: Policy Function



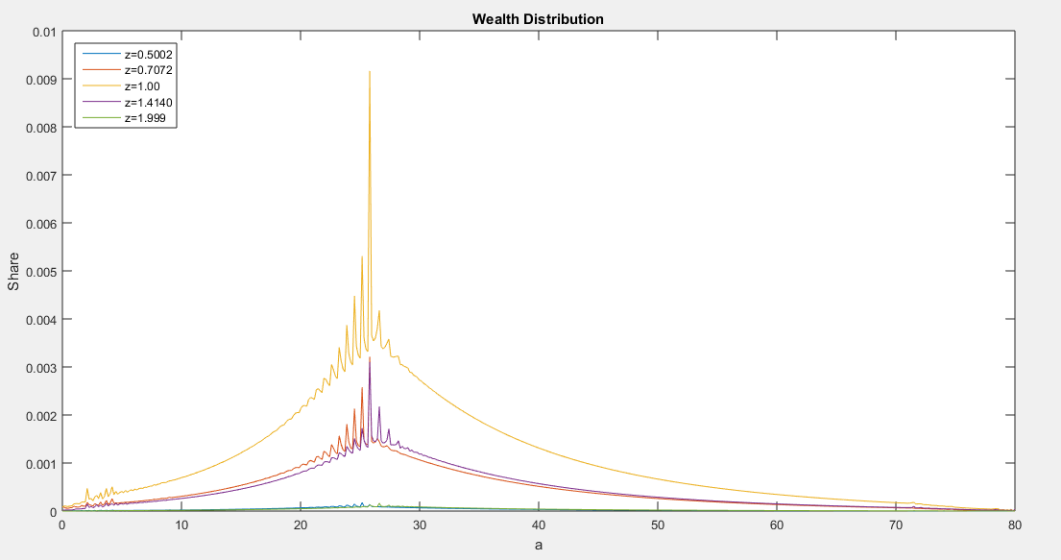
  - Figure (2) shows the Lorenz Curve and the Gini coefficient. As we can observe, the Gini for wealth is 0.24662.

Figure 2: Lorenz Curve and Gini



In addition, figure (3) shows the distribution of wealth. As we can observe, we a curve with more ' bell shape' in comparison with the Huggett model. Furthermore, this curve seems to have are less asymmetry or skewness now.

Figure 3: Lorenz Curve and Gini



– Alternative ways to solve for the value function.

For the interpolation methods I started with a reduced grid (a_num0) and then used the pice-wise polynomial to interpolate the guess value function to the complete grid (a_num0). Specifically, I followed the idea in Algorithm 4.2.1. in Heer and Maussner, and I tried to improve the code for the cubic spline interpolation as follows:

1. For a short grid (a_num0) I used the VFI method to get a discrete value function guess, v^1.
2. Using v^1 I constructed the pice-wise polynomial and optimize the Bellman equation between a_i and a_{i+1} using the Golden Section Search. I did this step for all the wealth states ($m = 5$).
3. Using this optimal value a^* (from the golden section search), I updated the polynomial functions.
4. Using the updated ('optimal') polynomial for the small grid, I constructed a new guess v^2 for the larger grid, a_num.
5. Using this guess I repeated the second step. Thus, the final results are a value function and a policy function for the large grid.

In addition, for the cubic spline I tried to adjust the policy function. This adjustment is because some 'optimal' values (optimal in terms of the golden section search) of a_i^* may be outside the range of original values on the grid. For instance, if I got a value of $a_i^* = 5.7$ and the grid is only with the numbers 5 and 6, I adjusted the policy function by saying that the number 5.7 is 30% of 5 and 70% of 6. Notice that this also represented

a modification of the iteration of the distribution.

Table (1) shows the runtime needed for each method. The policy function iteration makes a good improvement to the VFI. Additionally, the linear interpolation increases the speed by having a lower runtime, however this in exchange of precision. Finally, I did not find gains in term of runtime when using cubic spline interpolation. This may be related to the additional optimization step with the golden section search.

Table 1: Application of Different Methods to Solve VFI.

Method	run-time (s)
Value Function Iteration	176.6508
Policy Function Iteration	125.5012
Grid Search	72.0917
Interpolation - Linear	57.8396
Interpolation - Cubic Spline	23468.3401

A final comment about these methods. The convergence is very sensitive to different inputs in the code, such as the tolerance as criterion of convergence, and to the length of the grid a_num. Thus, the interpolations methods may spend hours stuck around the same K_guess when changing the precision or tolerance, or when using a different length for the grid. For instance, linear interpolation was tested for $a\_num = 500$ and $a\_num = 50$, getting faster results in comparison to the use of $a\_num = 12$.

lchanci1@binghamton.edu

Annex. Matlab Codes

- Central Code: Hw4_Chanci.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Binghamton University                        %
% PhD in Economics                             %
% ECON634 Advanced Macroeconomics              %
% Fall 2017                                    %
% Luis Chancí (lchanci1@binghamton.edu)        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all; close all; clc;

% OPTIONS (METHOD):
% 0 = VALUE FUNCTION ITERATION
% 1 = POLICY FUNCTION ITERATION
% 2 = GRID SEARCH
% 3 = INTERPOLATION - LINEAR
% 4 = INTERPOLATION - CUBIC SPLINE

   Option = 0 ;

% PARAMETERS
beta    = 0.99;      sigma   = 2.0;
alpha   = 1/3;       delta   = 0.025;
rho     = 0.5;       sigma_e = 0.2;
tol     = 0.05;

% DISCRETIZATION:
m                    = 5;
[z, PI]              = TAUCHEN(m, rho, sigma_e, 3);
z                    = exp(z');
[eig_vec, eig_val]   = eig(PI');
PI_ss                = bsxfun(@rdivide,eig_vec,sum(eig_vec));

% ASSETS, CAPITAL AND LABOR:
a_lo    = 0;
a_hi    = 80;
num_a   = 500;   % If want to see resuts for Opt=4 on the same day, use only 50.
num_a0  = 0.1*num_a;   % Initial Grid for interpolation
a       = linspace(a_lo, a_hi, num_a);
a0      = linspace(a_lo, a_hi, num_a0);
K_min   = 20; K_max = 40;
L       = (z*PI_ss(:,1));

e0 = 1; itera = 1; t0 = tic;
while (e0 >tol)        % (e0 >0.01)||(itera <=1000)
 K_guess = (K_min + K_max) / 2;
 r       = alpha*(K_guess ^ (alpha - 1))*(L ^ (1 - alpha))+(1-delta);
 w       = (1-alpha)*(K_guess ^ alpha)*(L^(-alpha));

 % I. VALUE FUNCTION (SOLUTION):

 if      Option == 0
         u           = Ch_U(r,w,a,z,sigma);
         v0          = zeros(m, num_a);
         [v0,g,idx]  = Ch_VFI(u,v0,beta,PI,a);
 elseif  Option == 1
         Ch_PFI
 elseif  Option == 2
         Ch_Grid_Search
 elseif  Option == 3
         Ch_Interpol_Linear
 elseif  Option == 4
         % (i) Initial guess (short grid).
         u             = Ch_U(r,w,a0,z,sigma);
```

```
        v0              = zeros(m, num_a0);
        [V0,G0,idx0]    = Ch_VFI(u,v0,beta,PI,a0);
        c0              = bsxfun(@plus, r*a0',permute(z,[1 3 2])*w);
        [V1,G1,VP]      = Ch_Interpol_Spline(V0,G0,idx0,c0,a0,PI,sigma,beta);

        % (ii) Interpolation for the complete grid
        u               = Ch_U(r,w,a,z,sigma);
        v_guess         = ppval(VP,a);
        [V2,G2,idx2]    = Ch_VFI(u,v_guess,beta,PI,a);
        c               = bsxfun(@plus, r*a',permute(z,[1 3 2])*w);
        [v0,g,vp]       = Ch_Interpol_Spline(V2,G2,idx2,c,a,PI,sigma,beta);
 end

% II. ITERATION OVER THE DISTRIBUTION
 Mu = ones(size(g))/numel(g);
 if Option <= 3
     Ch_Mu_standard
 else
     Ch_Mu_Spline
 end

% III. MARKETS CLEAR
 aggsav = sum(sum(Mu.*g));
 if (aggsav - K_guess)>0
    K_min = K_guess; elseif (aggsav - K_guess)<0; K_max = K_guess;
 end
 e0 = abs(aggsav - K_guess); itera = itera + 1;
 display (['K = ', num2str(K_guess)])
 display (['Aggregate desired wealth = ', num2str(aggsav)]);
end;  t = toc(t0);           % Now we can plot :)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- Value Function Iteration - Function: Ch_VFI.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Binghamton University                            %
% ECON634 Advanced Macroeconomics                  %
% Luis Chancí (lchanci1@binghamton.edu)            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This program does the Value Function Iteration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[v0,g,idx]=Ch_VFI(u,v0,beta,PI,a)
  num_a = length(a);
  e1 = 1;
    while e1 >.0001; %1e-06
          v = u + beta * repmat(permute((PI*v0),[3 2 1]),[num_a 1 1]);
          [vfn,idx] = max(v,[],2);
          e1  = max(max(abs(permute(vfn,[3 1 2]) - v0)));
          v0  = permute(vfn,[3 1 2]);
    end
    idx = permute(idx,[3 1 2]);
    g   = a(idx);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- Policy Function Iteration: Ch_PFI.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Binghamton University                            %
% ECON634 Advanced Macroeconomics                  %
% Luis Chancí (lchanci1@binghamton.edu)            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This script does the Policy Function Iteration%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 u      = Ch_U(r,w,a,z,sigma);
 v0     = zeros(m, num_a);
 e1 = 1;
   while e1 >1e-06
        v        = u + beta * repmat(permute(PI*v0,[3 2 1]),[num_a 1 1]);
    [vfn,idx] = max(v,[],2);
    e1         = max(max(abs(permute(vfn,[3 1 2]) - v0)));
    v0         = permute(vfn,[3 1 2]);
    idx        = permute(idx,[3 1 2]);
    g          = a(idx);
    Q          = makeQmatrix(idx, PI);
    c          = bsxfun(@plus, bsxfun(@minus,r*a,g),z'*w);
    ufn        = (c.^(1-sigma))./(1-sigma); ufn = ufn(:);
   W           = v0(:);
    for      j = 1:30
             V = ufn + beta*Q*W; W = V;
    end
   v0          = reshape(W,m,num_a);
   end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- Grid Search: Ch_Grid_Search.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Binghamton University                           %
% ECON634 Advanced Macroeconomics                 %
% Luis Chancí (lchanci1@binghamton.edu)           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This script conducts a grid search              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 u          = Ch_U(r,w,a,z,sigma);
 v0         = zeros(m, num_a);
 e1 = 1;
    while e1 > tol
          v           = u + beta * repmat(permute(PI*v0,[3 2 1]),[num_a 1 1]);
      [vfn,idx] = max(v,[],2);
      e1          = max(max(abs(permute(vfn,[3 1 2]) - v0)));
      v0          = permute(vfn,[3 1 2]);
   end;
 idx        = permute(idx,[3 1 2]);
 g          = a(idx);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- Interpolation - Linear: Ch_Interpol_Linear.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Binghamton University                           %
% ECON634 Advanced Macroeconomics                 %
% Luis Chancí (lchanci1@binghamton.edu)           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This script does the VFI and the linear         %
% interpolation method                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
u                = Ch_U(r,w,a0,z,sigma);
v0               = zeros(m, num_a0);
V                = zeros(m,num_a);

[v0,g0,idx0] = Ch_VFI(u,v0,beta,PI,a0);

for i=1:m
          V(i,:)     = interp1(a0, v0(i,:), a);
end
u                = Ch_U(r,w,a,z,sigma);
v                = u + beta * repmat(permute(PI*V,[3 2 1]),[num_a 1 1]);
[vfn,idx]        = max(v,[],2);
v0               = permute(vfn,[3 1 2]);
idx              = permute(idx,[3 1 2]);
g                = a(idx);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- Interpolation - Cubic Spline: Ch_Interpol_Spline.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Binghamton University                        %
% ECON634 Advanced Macroeconomics              %
% Luis Chancí (lchanci1@binghamton.edu)        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This program is for the interpolation using  %
% cubic spline method.                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [v0,G,v_polynm] = Ch_Interpol_Spline(v,g,idx,c,a,PI,sigma,beta)

 v_polynm = spline(a,v);
 num_a    = length(a);
 m        = length(PI);
 v0       = zeros(m, num_a);
 V        = zeros(m,num_a);
 G        = zeros(m,num_a);

 e1 = 1;
 while e1 >0.01 %1e-06
       for i = 1:m
           for j=1:num_a
               Bellman = @(x)((((c(j,1,i)-x).^(1-sigma))/(1 - sigma))...
                                 + beta*PI(i,:)*ppval(v_polynm,x));
               if idx(i,j) == 1
                  a_min = g(i,j);          a_max = a(idx(i,j)+1);
               elseif idx(i,j)== num_a
                  a_min = a(idx(i,j)-1); a_max = g(i,j);
               else
                  a_min = a(idx(i,j)-1); a_max = a(idx(i,j)+1);
               end
               % if isreal(Bellman(a_min))==1 && isreal(Bellman(a_max))==1
               a_min=0; if a_max > c(j,1,i); a_max = c(j,1,i); end
               [V(i,j),G(i,j)] = Ch_goldensectionsearch(Bellman,a_min,a_max);
               %end
           end
       end
        v_polynm = spline(a,V);
       e1        = max(max(abs(V - v0)));
       v0        = V;
 end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- Distribution μ: Ch_Mu_standard.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Binghamton University                            %
% ECON634 Advanced Macroeconomics                  %
% Luis Chancí (lchanci1@binghamton.edu)            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This script iterates over the distribution       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 e2 = 1;
 while e2>=1e-06;
        [emp_ind, a_ind, mass] = find(Mu);
        MuNew  = zeros(size(Mu));
        for ii = 1:length(emp_ind)
            apr_ind = idx(emp_ind(ii), a_ind(ii));
            MuNew (:, apr_ind) = MuNew (:, apr_ind) + ...
                  (PI(emp_ind(ii), :) * mass(ii))';
        end
        e2 = max(max(abs(MuNew  - Mu)));
        Mu = MuNew;
 end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- Distribution μ for cubic spline interpolation: Ch_Mu_Spline.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Binghamton University                            %
% ECON634 Advanced Macroeconomics                  %
% Luis Chancí (lchanci1@binghamton.edu)            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This script iterates over distributions by       %
% firstly adjusting the policy function.           %
% For instance, for 5.7: 30% of 5 and 70% of 6     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
idx1  = zeros(m,num_a,2); idx2 = idx1;
for i = 1:m
    for j = 1:num_a
        idx1(i,j,1) = find(a > g(i,j),1)-1;
        idx1(i,j,2) = find(a > g(i,j),1);
        idx2(i,j,1) = (g(i,j)-a(idx1(i,j,1)))/(a_hi/(num_a-1));
        idx2(i,j,2) = 1-(g(i,j)-a(idx1(i,j,1)))/(a_hi/(num_a-1));
    end
end

e2 = 1;
 while e2>=1e-06;
        [emp_ind, a_ind, mass] = find(Mu);
        MuNew  = zeros(size(Mu));
        for ii = 1:length(emp_ind)
            apr_idx1 = idx1(emp_ind(ii), a_ind(ii),:);
            apr_idx2 = idx2(emp_ind(ii), a_ind(ii),:);
            MuNew (:, apr_idx1(:,:,1)) = MuNew (:, apr_idx1(:,:,1)) + ...
                  apr_idx2(:,:,1)*(PI(emp_ind(ii), :) * mass(ii))';
            MuNew (:, apr_idx1(:,:,2)) = MuNew (:, apr_idx1(:,:,2)) + ...
                  apr_idx2(:,:,2)*(PI(emp_ind(ii), :) * mass(ii))';
        end
        e2 = max(max(abs(MuNew - Mu)));
        Mu = MuNew;
 end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- Utility Function: Ch_U.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Binghamton University                           %
% ECON634 Advanced Macroeconomics                 %
% Luis Chancí (lchanci1@binghamton.edu)           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This program is for the utility function        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function u = Ch_U(r,w,a,z,s)
 c       = bsxfun(@plus, bsxfun(@minus,r*a',a),permute(z,[1 3 2])*w);
 u       = (c.^(1-s))./(1-s);
 u(c<0) = -Inf;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- Modified Golden Section Search: Ch_goldensectionsearch.m

```
function [val, x_opt] = goldensectionsearch( f, a, b )
% Copyright (c) 2009, Katarzyna Zarnowiec, all rights reserved
f = @(x) -f(x);
epsilon=0.000001;               % accuracy value
iter= 50;                       % maximum number of iterations
tau=double((sqrt(5)-1)/2);      % golden proportion coefficient, around 0.618
k=0;                            % number of iterations
x1=a+(1-tau)*(b-a);             % computing x values
x2=a+tau*(b-a);
f_x1=f(x1);                     % computing values in x points
f_x2=f(x2);
while ((abs(b-a)>epsilon) && (k<iter))
    k=k+1;
    if(f_x1<f_x2)
        b=x2;
        x2=x1;
        x1=a+(1-tau)*(b-a);
        f_x1=f(x1);
        f_x2=f(x2);
    else
        a=x1;
        x1=x2;
        x2=a+tau*(b-a);
        f_x1=f(x1);
        f_x2=f(x2);
    end
    k=k+1;
end
if(f_x1<f_x2)
    x_opt = x1;
    val   = -f_x1;
else
    x_opt = x2;
    val   = -f_x2;
end
end
```