## Table of Contents



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MATLAB Code: HJB_Huggett_GE
%
% Author: Muhammad (Modified section code from Kiyea)
% Date: Nov 7, 2024
%
% Description:
% This MATLAB script solves the general equilibrium of the Huggett model,
% finding the equilibrium interest rate that clears the bond market.
%
% Reference: Huggett_equilibrium_iterate.m by Benjamin Moll
%
% Notes:
% - CRRA utility function: U(c) = (c^(1-sigma))/(1-sigma)
% - Elasticity of intertemporal substitution (sigma): 2
% - Discount rate (rho): 0.05
% - Income: z = [z_u, z_e] = [0.1, 0.2];
% - Lambda: la = [la_u, la_e] = [1.2, 1.2];
% - Discrete grid of asset levels (a): -0.15 to 5
% - Borrowing constraint: a>=-0.15
% - Delta = 1000; (Can be arbitrarily large in implicit method)
%
% Code Structure:
% 1. DEFINE PARAMETERS
% 2. INITIALIZE GRID POINTS
% 3. PRE-ITERATION INITIALIZATION
% 4. VALUE FUNCTION ITERATION
% 5. KF EQUATION
% 6. GRAPHS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
close all;
clc;
```

# 1. DEFINE PARAMETERS

```
addpath(genpath('/Users/muhammadbashir/GitHub/MuhammadCourses/
DynamicProgramming2024/ProblemSetSolutions/PS4/GE/'))
p = define_parameters_GE();
```

# 2. INITIALIZE GRID POINTS

```
a = linspace(p.amin, p.amax, p.I)';
da = (p.amax-p.amin)/(p.I-1);

aa = [a, a]; % I*2 matrix

        % %% 2-2. INITIALIZE GRID POINTS FOR INTEREST RATES
        %
        % rgrid = linspace(p.rmin, p.rmax, p.Ir)';
```

# 3. PRE-ITERATION INITIALIZATION

```
% 3-1. Construct the forward and backward differential operator
% Df such that Df*V=dVf and Db such that Db*V=dVb

    Df = zeros(p.I, p.I);
    for i = 1:p.I-1
        Df(i,i) = -1/da; Df(i,i+1) = 1/da;
    end
    Df = sparse(Df);

    Db = zeros(p.I, p.I);
    for i = 2:p.I
        Db(i,i-1) = -1/da; Db(i,i) = 1/da;
    end
    Db = sparse(Db);

% 3-2. Construct A_switch matrix

    A_switch = [speye(p.I).*(-p.lambda(1)), speye(p.I).*p.lambda(1);
                speye(p.I).*p.lambda(2), speye(p.I).*(-p.lambda(2))];

%A_switch = zeros(2*I, 2*I);
%for i=1:I
%    A_switch(i,i) = -lambda(1);
%    A_switch(i,i+I) = lambda(1);
%    A_switch(i+I,i) = lambda(2);
%    A_switch(i+I,i+I) = -lambda(2);
%end
```

# 3-3. Guess an initial value of the interest rate

```
r0 = 0.03;
r_min = 0.01;
r_max = 0.04;
```

# 3-4. Guess an initial value of the value function

```
zz = ones(p.I, 1).*p.zz; % I*2 matrix

% The value function of "staying put"
r = r0;

v0 = p.u(zz + r.*aa)./p.rho;
V = v0;
```

# 4. VALUE FUNCTION ITERATION

```
for nr=1:p.Nr

    r_r(nr) = r;
    rmin_r(nr) = r_min;
    rmax_r(nr) = r_max;

    % Use the value function solution from the previous interest rate
iteration
    % as the initial guess for the next iteration
    if nr>1
       v0 = V_r(:,:,nr-1);
       V = v0;
    end
```

# 4. VALUE FUNCTION ITERATION

```
for n=1:p.maxit

    % 4-1. Compute the derivative of the value function
    dVf = Df*V;
    dVb = Db*V;

    % 4-2. Boundary conditions
    dVb(1,:) = p.mu(zz(1,:) + r.*aa(1,:)); % a>=a_min is enforced (borrowing
constraint)
    dVf(end,:) = p.mu(zz(end,:) + r.*aa(end,:)); % a<=a_max is enforced which
helps stability of the algorithm

    I_concave = dVb > dVf; % indicator whether value function is concave
(problems arise if this is not the case)

    % 4-3. Compute the optimal consumption
    cf = p.inv_mu(dVf);
```

```
    cb = p.inv_mu(dVb);

    % 4-4. Compute the optimal savings
    sf = zz + r.*aa - cf;
    sb = zz + r.*aa - cb;

    % 4-5. Upwind scheme
    If = sf>0;
    Ib = sb<0;
    I0 = 1-If-Ib;
    dV0 = p.mu(zz + r.*aa); % If sf<=0<=sb, set s=0

    dV_upwind = If.*dVf + Ib.*dVb + I0.*dV0;

    c = p.inv_mu(dV_upwind);

    % 4-6. Update value function:
    % Vj^(n+1) = [(rho + 1/Delta)*I - (Sj^n*Dj^n+A_switch)]^(-1)*[u(cj^n) +
1/Delta*Vj^n]

    V_stacked = V(:); % 2I*1 matrix
    c_stacked = c(:); % 2I*1 matrix

    % A = SD
    SD_u = spdiags(If(:,1).*sf(:,1), 0, p.I, p.I)*Df +
spdiags(Ib(:,1).*sb(:,1), 0, p.I, p.I)*Db; % I*I matrix
    SD_e = spdiags(If(:,2).*sf(:,2), 0, p.I, p.I)*Df +
spdiags(Ib(:,2).*sb(:,2), 0, p.I, p.I)*Db; % I*I matrix
    SD = [SD_u, sparse(p.I, p.I);
         sparse(p.I, p.I), SD_e]; % 2I*2I matrix

    % P = A + A_switch
    P = SD + A_switch;

    % B = [(rho + 1/Delta)*I - P]
    B = (p.rho + 1/p.Delta)*speye(2*p.I) - P;

    % b = u(c) + 1/Delta*V
    b = p.u(c_stacked) + (1/p.Delta)*V_stacked;

    % V = B\b;
    V_update = B\b; % 2I*1 matrix
    V_change = V_update - V_stacked;
    V = reshape(V_update, p.I, 2); % I*2 matrix

    % 3-6. Convergence criterion
    dist(n) = max(abs(V_change));
    if dist(n)<p.tol
       disp('Value function converged. Iteration = ')
       disp(n)
       break
    end
end
```

```
toc;
```

*Value function converged. Iteration =*
*9*

*Elapsed time is 13.659019 seconds.*

*Value function converged. Iteration =*
*5*

*Elapsed time is 13.716215 seconds.*

*Value function converged. Iteration =*
*5*

*Elapsed time is 13.756018 seconds.*

*Value function converged. Iteration =*
*4*

*Elapsed time is 13.778938 seconds.*

*Value function converged. Iteration =*
*4*

*Elapsed time is 13.801735 seconds.*

# 5. KF EQUATION

```
% 5-1. Solve for 0=gdot=P'*g

PT = P';
gdot_stacked = zeros(2*p.I,1);

% need to fix one value, otherwise matrix is singular
i_fix = 1;
gdot_stacked(i_fix)=.1;

row_fix = [zeros(1,i_fix-1),1,zeros(1,2*p.I-i_fix)];
AT(i_fix,:) = row_fix;

g_stacked = PT\gdot_stacked;

% 5-2. Normalization

g_sum = g_stacked'*ones(2*p.I,1)*da;
g_stacked = g_stacked./g_sum;

% 5-3. Reshape

gg = reshape(g_stacked, p.I, 2);
```

# 5-4. COMPUTE VARIABLES FOR A GIVEN r_r(nr)

Notes: Each matrix has dimensions p.I*2(u,e)*nr

```
g_r(:,:,nr) = gg;
adot(:,:,nr) = zz + r.*aa - c;
V_r(:,:,nr) = V;
dV_r(:,:,nr) = dV_upwind;
c_r(:,:,nr) = c;

S(nr) = gg(:,1)'*a*da + gg(:,2)'*a*da;
```

# 5-5. UPDATE INTEREST RATE

```
if nr == 1
    % Store the initial guess
    r_old = r;
    S_old = S(nr);
    % Update the interest rate using bisection method for the first
iteration
    if S(nr) > 0
    r_max = r;
    r = 0.5 * (r_min + r_max);
    elseif S(nr) < 0
    r_min = r;
    r = 0.5 * (r_min + r_max);
    end
else
    % Use Newton's method for subsequent iterations
    dr = (S(nr) - S_old) / (r - r_old);
    r_old = r;
    S_old = S(nr);
    r = r - S(nr) / dr;
end

% Check if the new interest rate is within bounds
if r < r_min || r > r_max
    r = 0.5 * (r_min + r_max);
end

% Check for convergence
if abs(S(nr)) < p.tol_S
    disp('Equilibrium Found, Interest rate =')
    disp(r)
    break
end
```

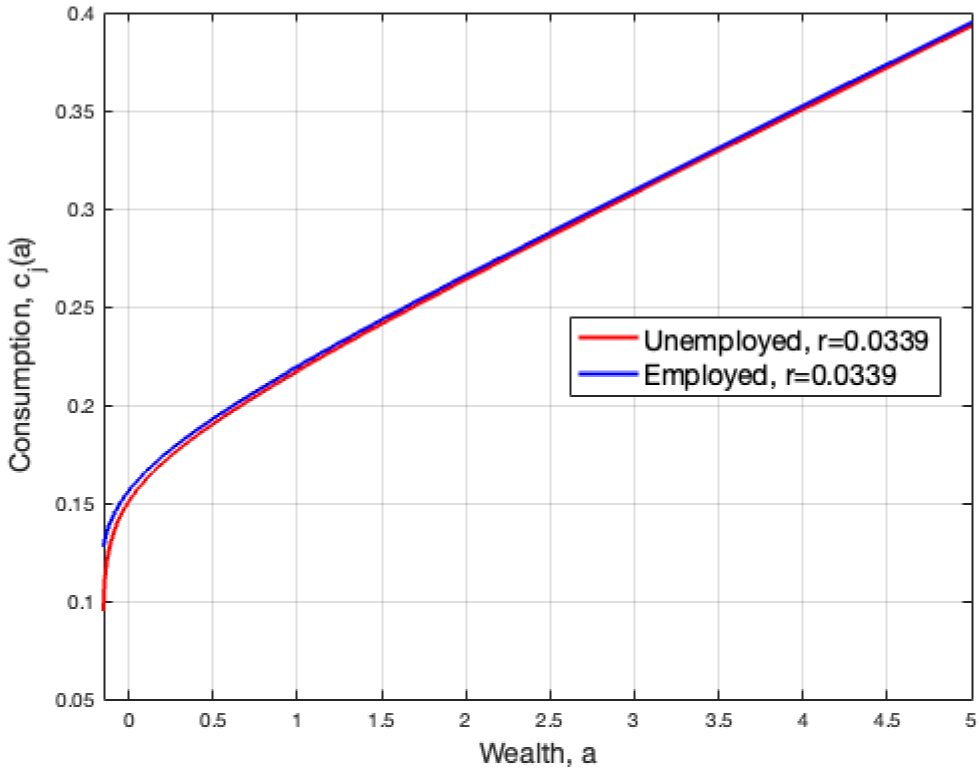
```
Equilibrium Found, Interest rate =
    0.0339

Algorithm converged
```

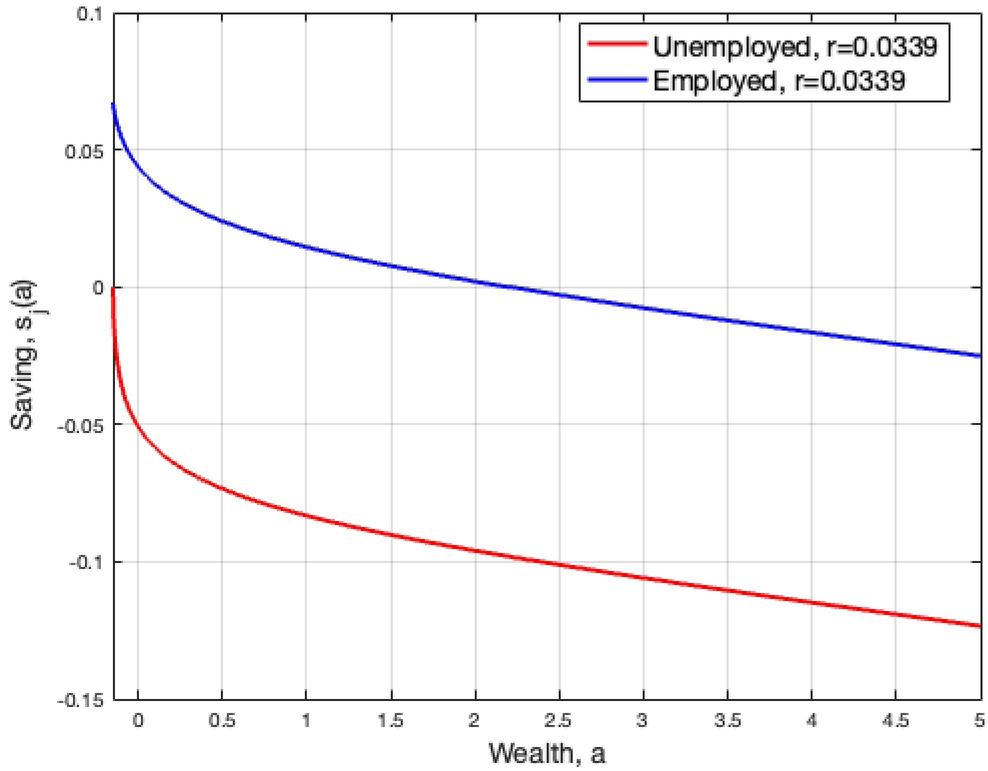
```
end

disp("Algorithm converged")
```

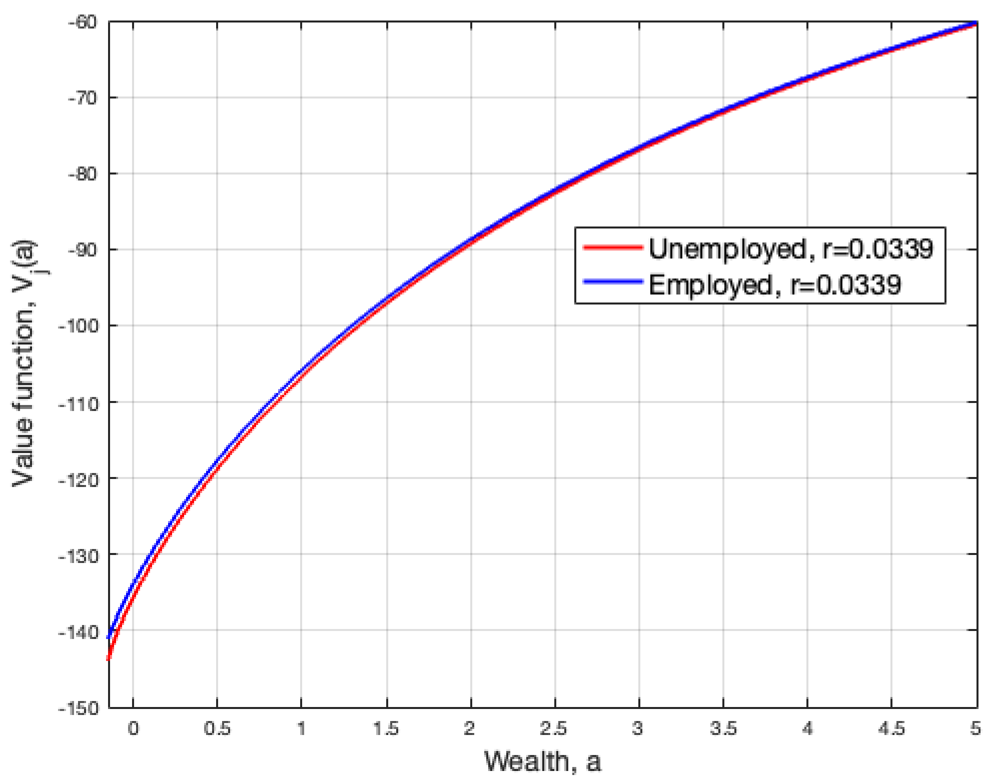
# 6. GRAPHS

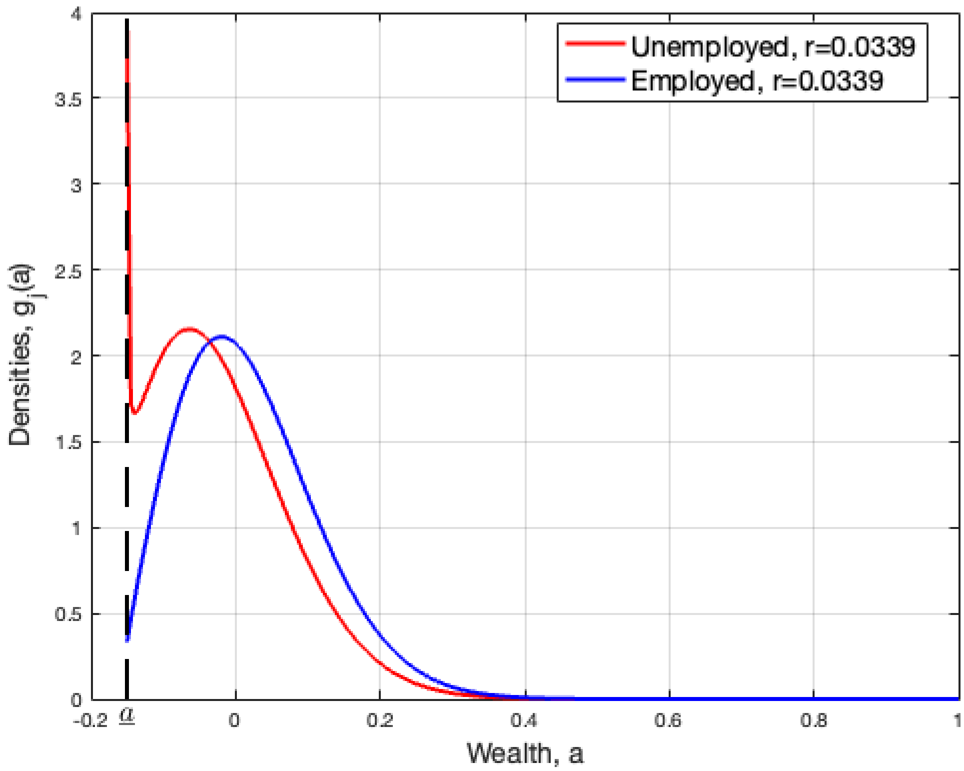
```
% 6-1. Optimal consumption
figure;
set(gca, 'FontSize', 18)
plot(a, c_r(:,1,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'r')
hold on
plot(a, c_r(:,2,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'b')
hold off
grid
xlabel('Wealth, a','FontSize', 14)
ylabel('Consumption, c_j(a)','FontSize', 14)
xlim([p.amin p.amax])
legend(sprintf('Unemployed, r=%.4f', r), ...
       sprintf('Employed, r=%.4f', r), 'Location', 'best', 'FontSize', 14)

% 6-2. Optimal savings
figure;

set(gca, 'FontSize', 18)
plot(a, adot(:,1,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'r')
hold on
plot(a, adot(:,2,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'b')
hold off
grid
xlabel('Wealth, a', 'FontSize', 14)
ylabel('Saving, s_j(a)', 'FontSize', 14)
xlim([p.amin p.amax])
legend(sprintf('Unemployed, r=%.4f', r), ...
       sprintf('Employed, r=%.4f', r), 'Location', 'best', 'FontSize', 14)

% 6-3. Value function
figure;

set(gca, 'FontSize', 18)
plot(a, V_r(:,1,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'r')
hold on
plot(a, V_r(:,2,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'b')
hold off
grid
xlabel('Wealth, a', 'FontSize', 14)
ylabel('Value function, V_j(a)', 'FontSize', 14)
xlim([p.amin p.amax])
legend(sprintf('Unemployed, r=%.4f', r), ...
       sprintf('Employed, r=%.4f', r), 'Location', 'best', 'FontSize', 14)

% 6-4. Wealth distribution
figure;
```

```
set(gca, 'FontSize', 14)
plot(a, g_r(:,1,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'r')
hold on
plot(a, g_r(:,2,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'b')
hold off
grid
xlabel('Wealth, a', 'FontSize', 14)
ylabel('Densities, g_j(a)', 'FontSize', 14)
yy = get(gca, 'yLim');
hold on
plot([p.amin, p.amin], yy, '--k', 'LineWidth', 2)
hold off
text(-0.15, yy(1)-0.02*(yy(2) - yy(1)), '$\underline{a}$',
'HorizontalAlignment', 'center', 'FontSize', 15, 'Interpreter', 'latex')
xlim([-0.2 1])
legend(sprintf('Unemployed, r=%.4f', r), ...
       sprintf('Employed, r=%.4f', r), 'Location', 'best', 'FontSize', 14)
```




Unemployed, r=0.0339
Employed, r=0.0339
Saving, $s_j(a)$
Wealth, a
0.1
0.05
0
-0.05
-0.1
-0.15
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


Unemployed, r=0.0339
Employed, r=0.0339
Value function, $V_j(a)$
Wealth, a
-60
-70
-80
-90
-100
-110
-120
-130
-140
-150
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5




*Published with MATLAB® R2024b*