Coding Exercise 4 Solution: Solving the General Equilibrium of the Huggett Model

ECON 202A

November 12, 2024

1. Optimal Consumption

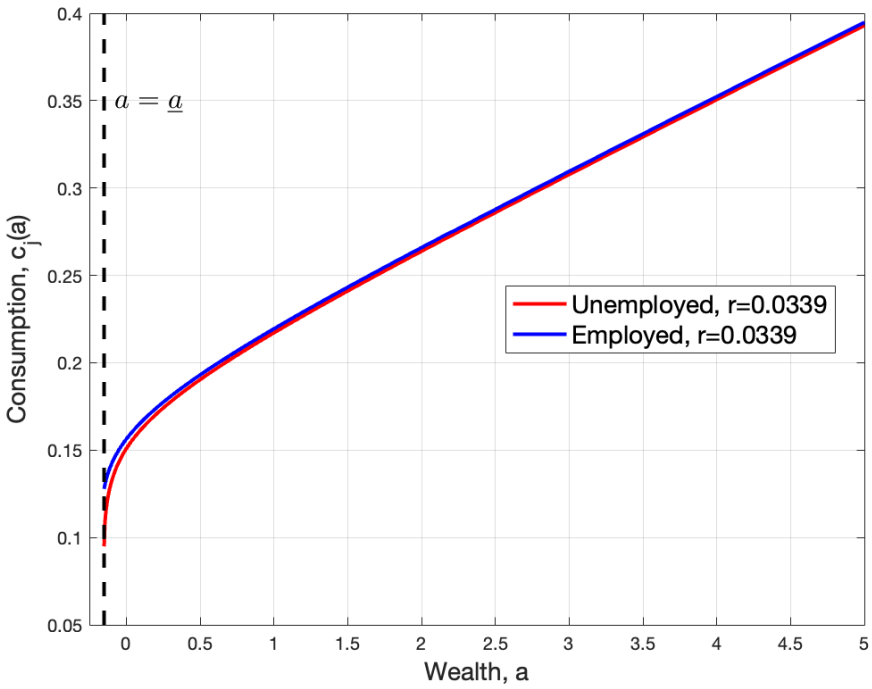

Figure 1: Optimal Consumption Policy for Employed and Unemployed States

For individuals close to the borrowing constraint, optimal consumption is lower, especially for the unemployed. This is because unemployed individuals have reduced income and limited access to credit, which restricts their ability to smooth consumption over time.

2. Optimal Savings

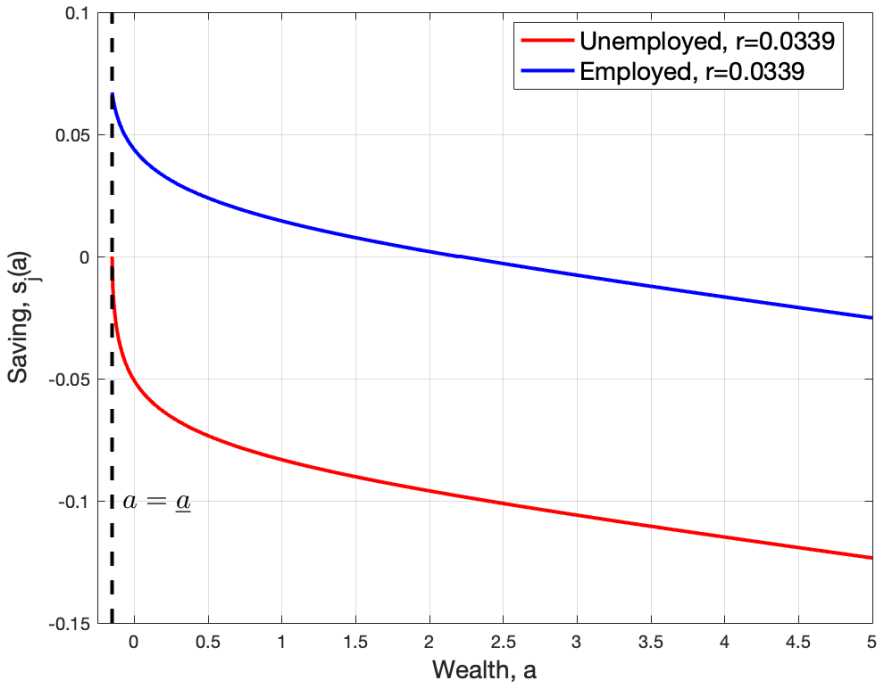

Figure 2: Optimal Savings Policy for Employed and Unemployed States

The savings function indicates that employed individuals are more likely to save (positive savings) as their wealth increases, while unemployed individuals typically experience dissaving (negative savings) due to their lower income. At lower asset levels, unemployed individuals dissave at a faster rate to fund their consumption, which brings them closer to the borrowing constraint. Employed individuals, in contrast, exhibit precautionary savings behavior, especially at lower asset levels, as they aim to build a financial buffer against potential future income shocks.

3. Wealth Distribution

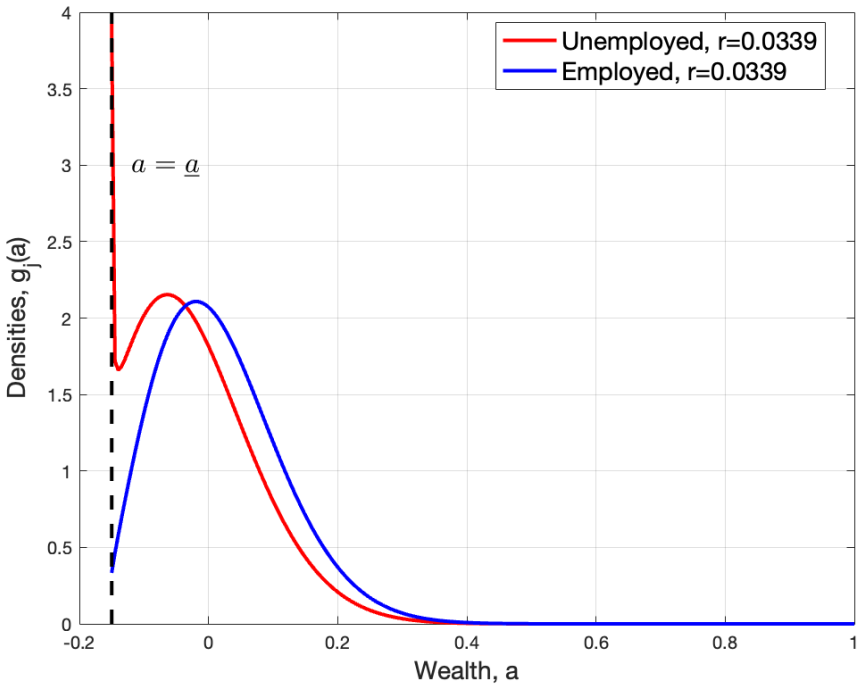

Figure 3: Wealth Distribution for Employed and Unemployed States

The wealth distribution reveals a higher density of unemployed individuals near the borrowing constraint, indicated by the sharp peak at low wealth levels. This suggests that unemployed individuals are more likely to have low or negative wealth due to limited income and frequent dissaving. Employed individuals, in contrast, exhibit a more dispersed distribution across higher wealth levels, reflecting their ability to save and accumulate wealth over time. The difference in wealth distributions between employed and unemployed individuals highlights the long-term impact of employment status and borrowing constraint on wealth accumulation and financial stability.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MATLAB Code: HJB_Huggett_GE_Newton
%
% Author: Kiyea Jin
% 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 using Newton's met
%
% 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
% NEWTON'S METHOD
% 6. GRAPHS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
close all;
clc;

%% 1. DEFINE PARAMETERS

p = define_parameters_GE();

%% NEWTON'S METHOD

% Guess an initial value of the interest rate

    r0 = 0.03;

% Use fsolve to find the equilibrium interest rate that makes S(r)=0
% Note: X = fsolve(FUN, X0, options) starts at the matrix X0 and tries to solve the eq
% Set OPTIONS = optimoptions('fsolve','Algorithm','trust-region'), and then pass OPTIO

    S = @(r) stationary(r, p);

    options = optimoptions('fsolve', 'TolFun', p.tol, 'Display', 'iter');
    r_equilibrium = fsolve(S, r0, options);

% Rerun stationary with the correct interest rate
    [~, ss] = stationary(r_equilibrium, p);
```

```
%% 6. GRAPHS

% 6-1. Optimal consumption

set(gca, 'FontSize', 18)
plot(ss.a, ss.c(:,1), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'r')
hold on
plot(ss.a, ss.c(:,2), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'b')
hold on
yy = get(gca, 'yLim');
plot([p.amin, p.amin], yy, '--k', 'LineWidth', 2)
hold off
grid
xlabel('Wealth, a','FontSize', 14)
ylabel('Consumption, c_j(a)','FontSize', 14)
xlim([p.amin-0.1 p.amax])
legend(sprintf('Unemployed, r=%.4f', r_equilibrium), ...
       sprintf('Employed, r=%.4f', r_equilibrium), 'Location', 'best', 'FontSize', 14)
text(-0.08,0.35,'$a=\underline{a}$','FontSize',16,'interpreter','latex')

% 6-2. Optimal savings

set(gca, 'FontSize', 18)
plot(ss.a, ss.adot(:,1), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'r')
hold on
plot(ss.a, ss.adot(:,2), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'b')
hold on
yy = get(gca, 'yLim');
plot([p.amin, p.amin], yy, '--k', 'LineWidth', 2)
hold off
grid
xlabel('Wealth, a', 'FontSize', 14)
ylabel('Saving, s_j(a)', 'FontSize', 14)
xlim([p.amin-0.1 p.amax])
legend(sprintf('Unemployed, r=%.4f', r_equilibrium), ...
       sprintf('Employed, r=%.4f', r_equilibrium), 'Location', 'best', 'FontSize', 14)
text(-0.08,-0.1,'$a=\underline{a}$','FontSize',16,'interpreter','latex')

% 6-3. Wealth distribution

set(gca, 'FontSize', 14)
plot(ss.a, ss.gg(:,1), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'r')
hold on
plot(ss.a, ss.gg(:,2), 'LineWidth', 2, 'LineStyle', '-', 'Color', 'b')
hold on
yy = get(gca, 'yLim');
plot([p.amin, p.amin], yy, '--k', 'LineWidth', 2)
hold off
grid
xlabel('Wealth, a', 'FontSize', 14)
ylabel('Densities, g_j(a)', 'FontSize', 14)
yy = get(gca, 'yLim');
xlim([p.amin-0.05 1])
```

```
legend(sprintf('Unemployed, r=%.4f', r_equilibrium), ...
       sprintf('Employed, r=%.4f', r_equilibrium), 'Location', 'best', 'FontSize', 14)
text(-0.12,3,'$a=\underline{a}$','FontSize',16,'interpreter','latex')
```

```
function [S, ss] = stationary(r0, p)

% This function solves the HJB equation and KF equation of the Huggett model
% for given interest rates r0.

%% 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

%% 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))];

% 3-3. 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 n=1:p.maxit
```

```
V = v;
V_u = V(:,1);
V_e = V(:,2);

% 4-1. Compute the derivative of the value function
dVf = [Df*V_u, Df*V_e];
dVb = [Db*V_u, Db*V_e];

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

I_concave = dVb > dVf; %indicator whether value function is concave (problems aris

% 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);
s = zz + r.*aa - c;

% 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

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

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

% P = A + A_switch
P = A + 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;

%% 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);

%% OUTPUT

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

ss.a = a;
ss.gg = gg;
ss.adot = zz + r.*aa - c;
ss.V = V;
ss.dV = dV_upwind;
ss.c = c;

end
```

```
function p = define_parameters_GE()

% This function defines the parameters needed for the Huggett_GE.m script

%% Economic Parameters

    % Relative risk aversion
    p.gamma = 2;

    % Discount rate
    p.rho = 0.05;

    %% WE NO LONGER ASSUME EXOGENOUS INTEREST RATE
    % Exogenous interest rate
    % p.r = 0.035;

    % Income process
    p.z_u = 0.1;
    p.z_e = 0.2;
    p.zz = [p.z_u, p.z_e];

    % Probability density
    p.lambda_u = 1.2;
    p.lambda_e = 1.2;
    p.lambda = [p.lambda_u, p.lambda_e];

%% Economic Functions

    % Utility funtion
    p.u = @(c) c.^(1-p.gamma)/(1-p.gamma);

    % Marginal utility function
    p.mu = @(c) c.^(-p.gamma);

    % FOC: mu(c)=dV -> c=inv_mu(dV)
    p.inv_mu = @(dV) dV.^(-1/p.gamma);

%% Grid Parmaters

    p.amin = -0.15;
    p.amax = 5;

    % The number of grid points
    p.I = 1000;

    % % WE NO LONGER CONSTRUCT GRID POINTS FOR INTEREST RATES
    % % Grid parameters for interest rate
    % p.rmin = -0.05;
    % p.rmax = 0.04;
    % p.Ir = 20;

%% Tuning parameters
```

```
    % Step size: can be arbitrarily large in implicit method
    p.Delta = 1000;

    % The maximum number of value function iterations
    p.maxit = 100;

    % Tolerance for value function iterations
    p.tol = 10^(-6);

    %% TUNING PARAMETERS FOR INTEREST RATE ITERATION

    % The maximum number of interest rate iterations
    p.Nr = 40;

    % Tolerance for interest rate iterations
    p.tol_S = 10^(-6);

end
```