3. Relative to $r = .35$, the distributions for both the employed & unemployed groups shift rightwards. This change reflects the fact that individuals save more in response to higher interest rates.

4. From $\lambda_e = \lambda_u = 1/3$, the steady state employment & unemployment rates are .5. Equilibrium aggregate labor is 1.5.

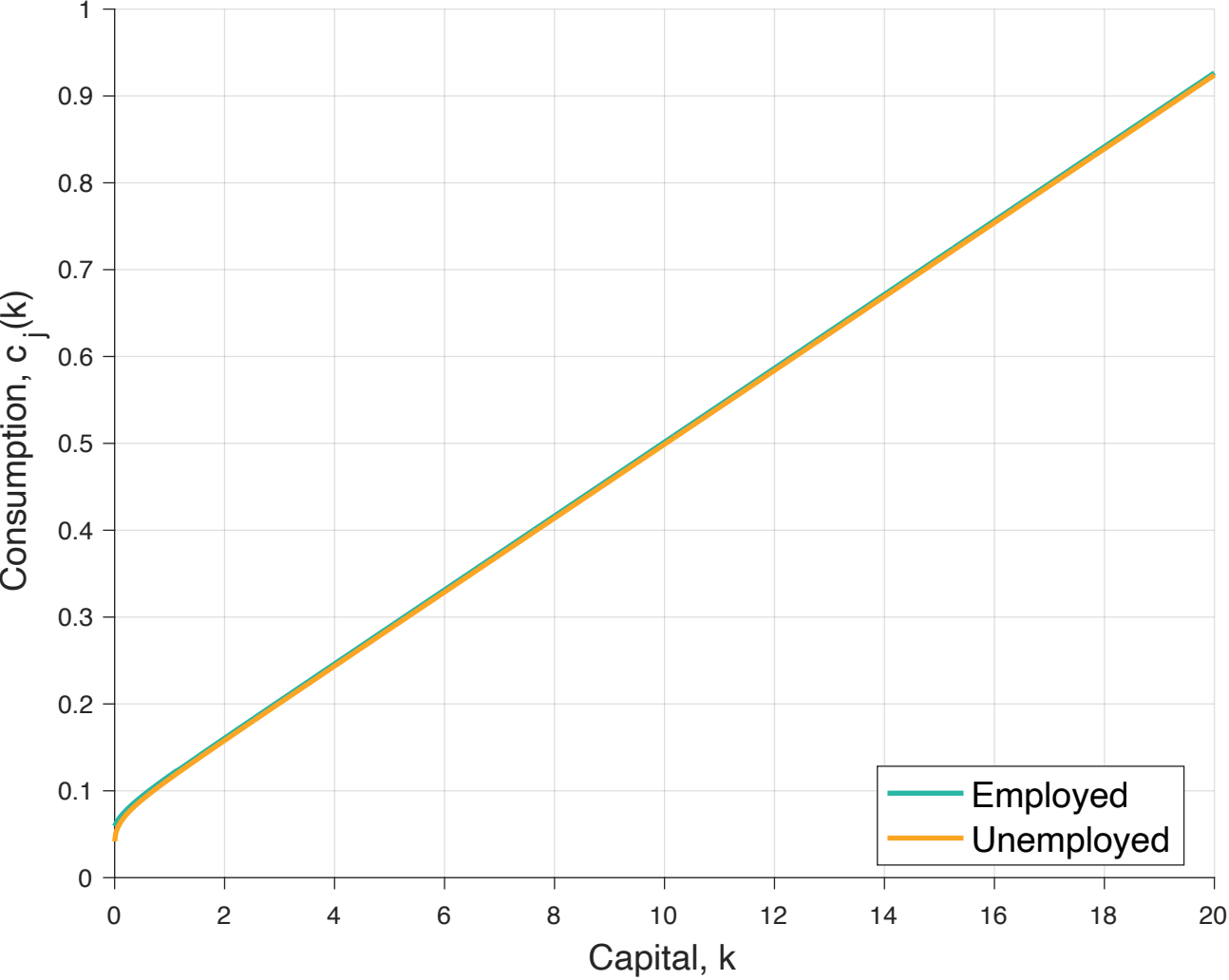
5. The stationary equilibrium interest rate is .045.

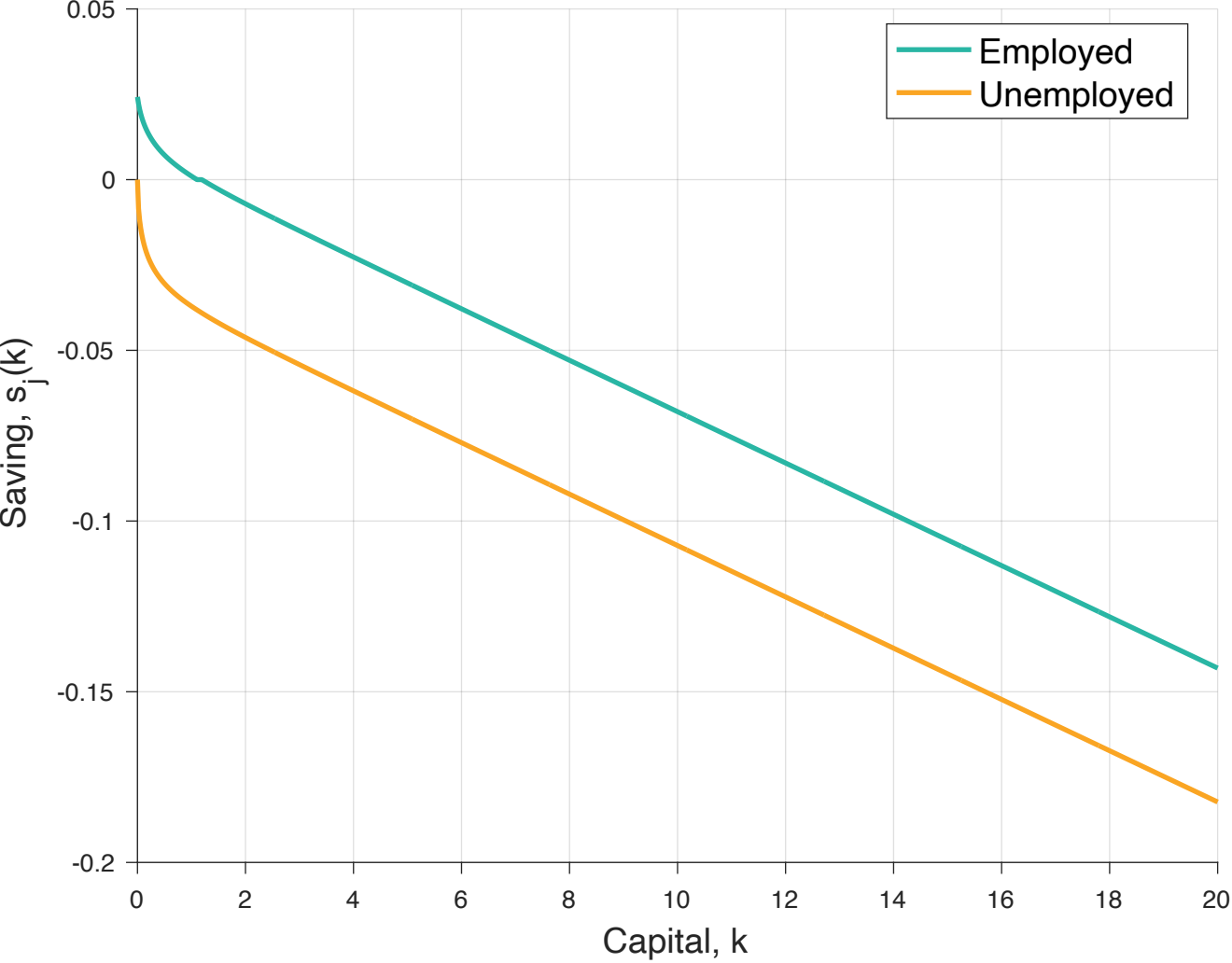
6. We see the unemployed group has higher consumption & savings at each value of capital, which aligns with economic intuition. Additionally, the unemployed group does not save at any capital level. This savings behavior reflects agents' desire to consumption smooth. Given this equilibrium interest rate (.045) is larger than in the partial equilibrium models discussed above,

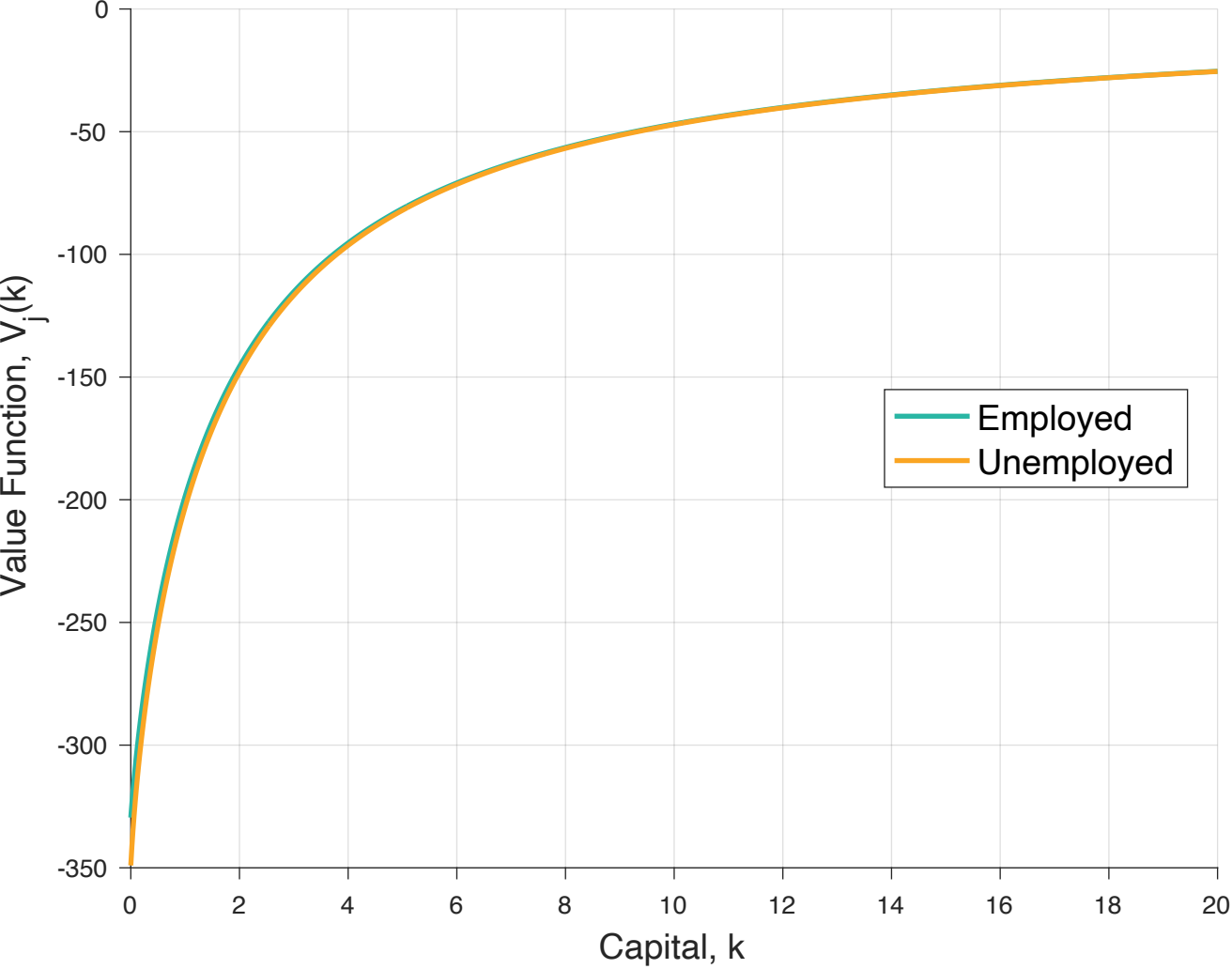
we see that the capital distribution is shifted even more rightward here.

8. We see that interest rates, wages, & the stock of capital all decrease in response to a negative TFP shock. Wages decrease since the marginal product of labor decreases. The negative TFP shock causes capital to decrease for two reasons: i. lower TFP decreases the marginal product of capital & ii. the decrease in interest rates prompt individuals to disinvest. All return to their steady state values eventually.

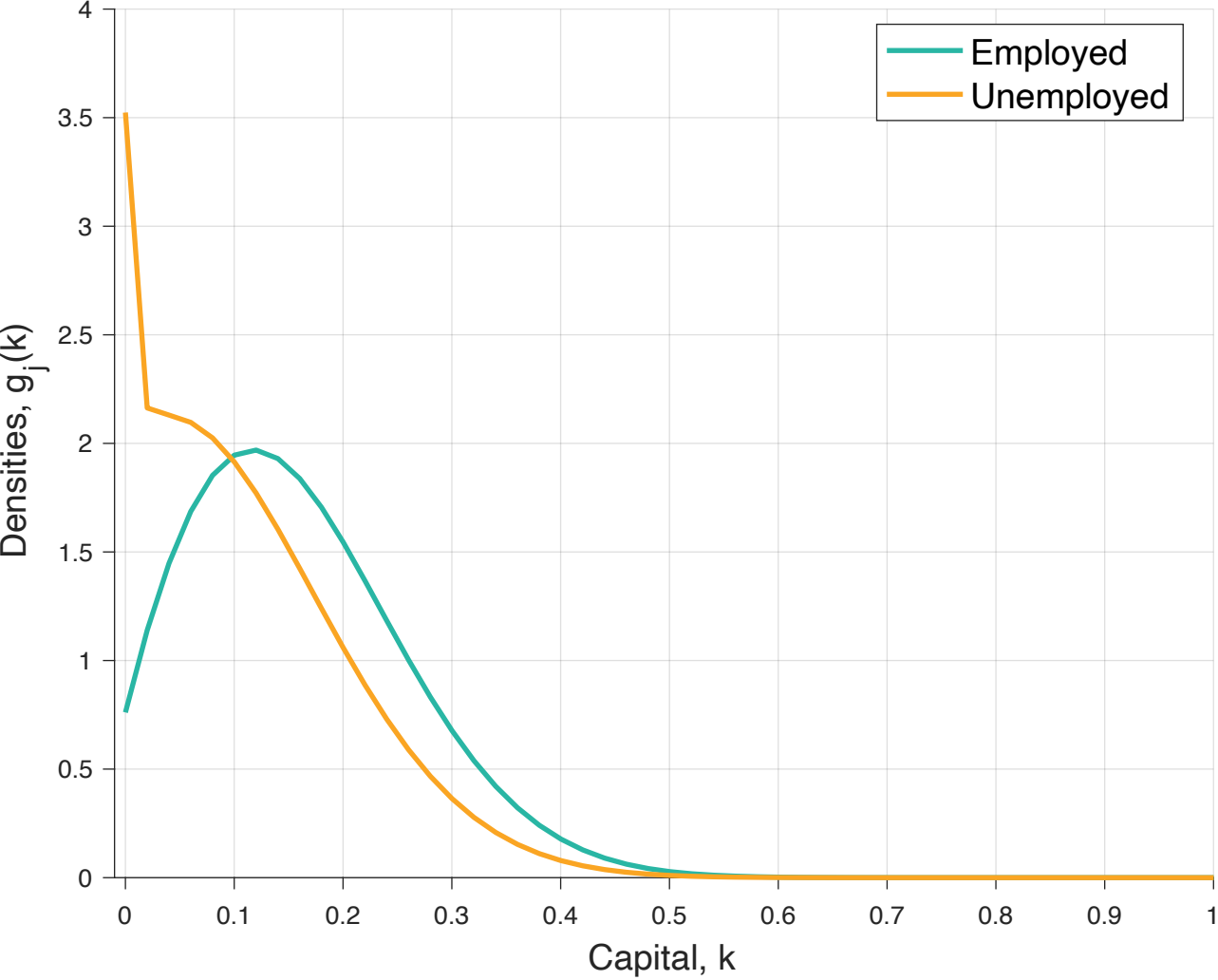
Question 1


Consumption, $c_j(k)$
Capital, k
Employed
Unemployed
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
2
4
6
8
10
12
14
16
18
20

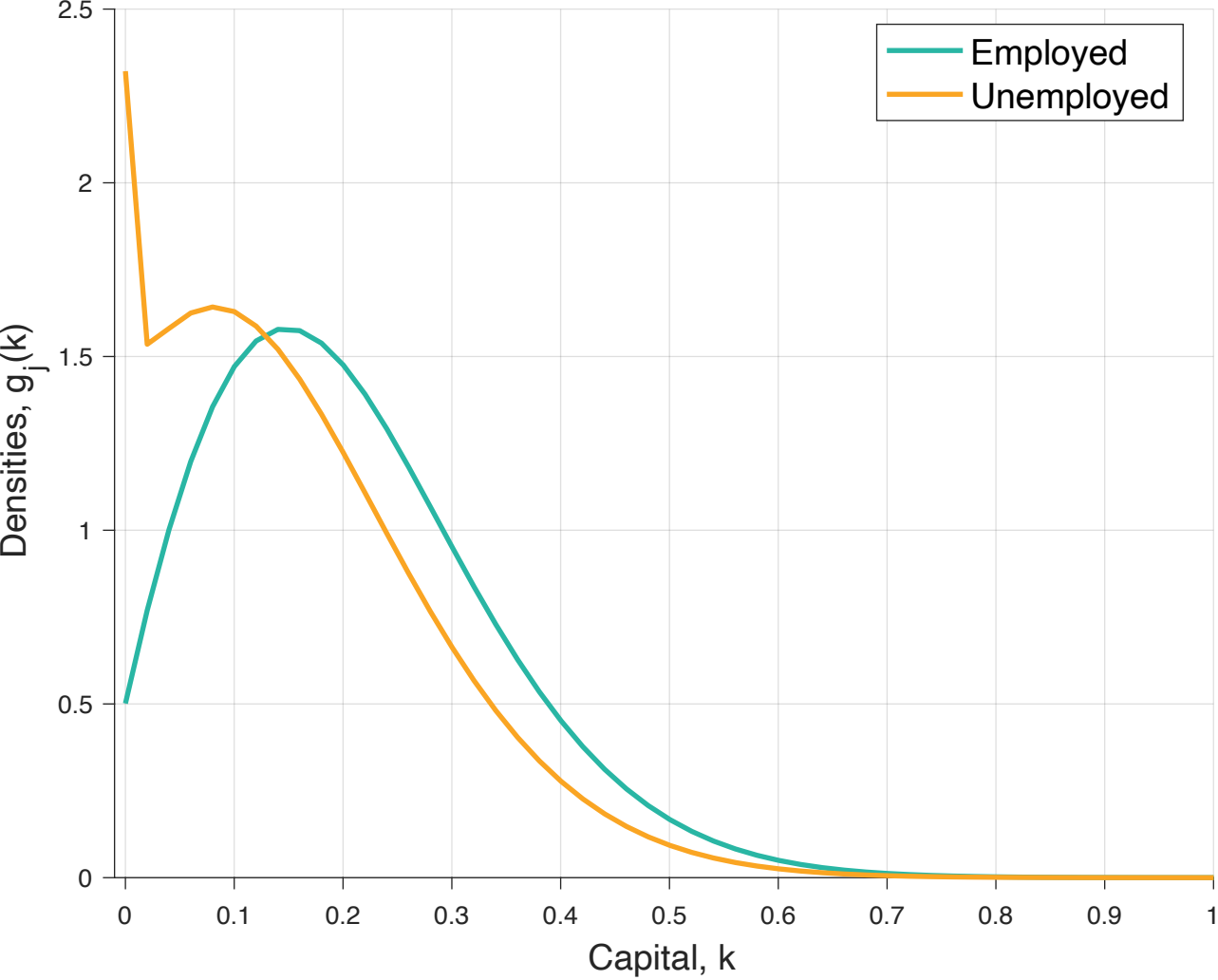

Employed
Unemployed
0.05
0
-0.05
-0.1
-0.15
-0.2
Saving, $s_j(k)$
0
2
4
6
8
10
12
14
16
18
20
Capital, k


0
-50
-100
-150
-200
-250
-300
-350
Value Function, $V_j(k)$
0
2
4
6
8
10
12
14
16
18
20
Capital, k
Employed
Unemployed

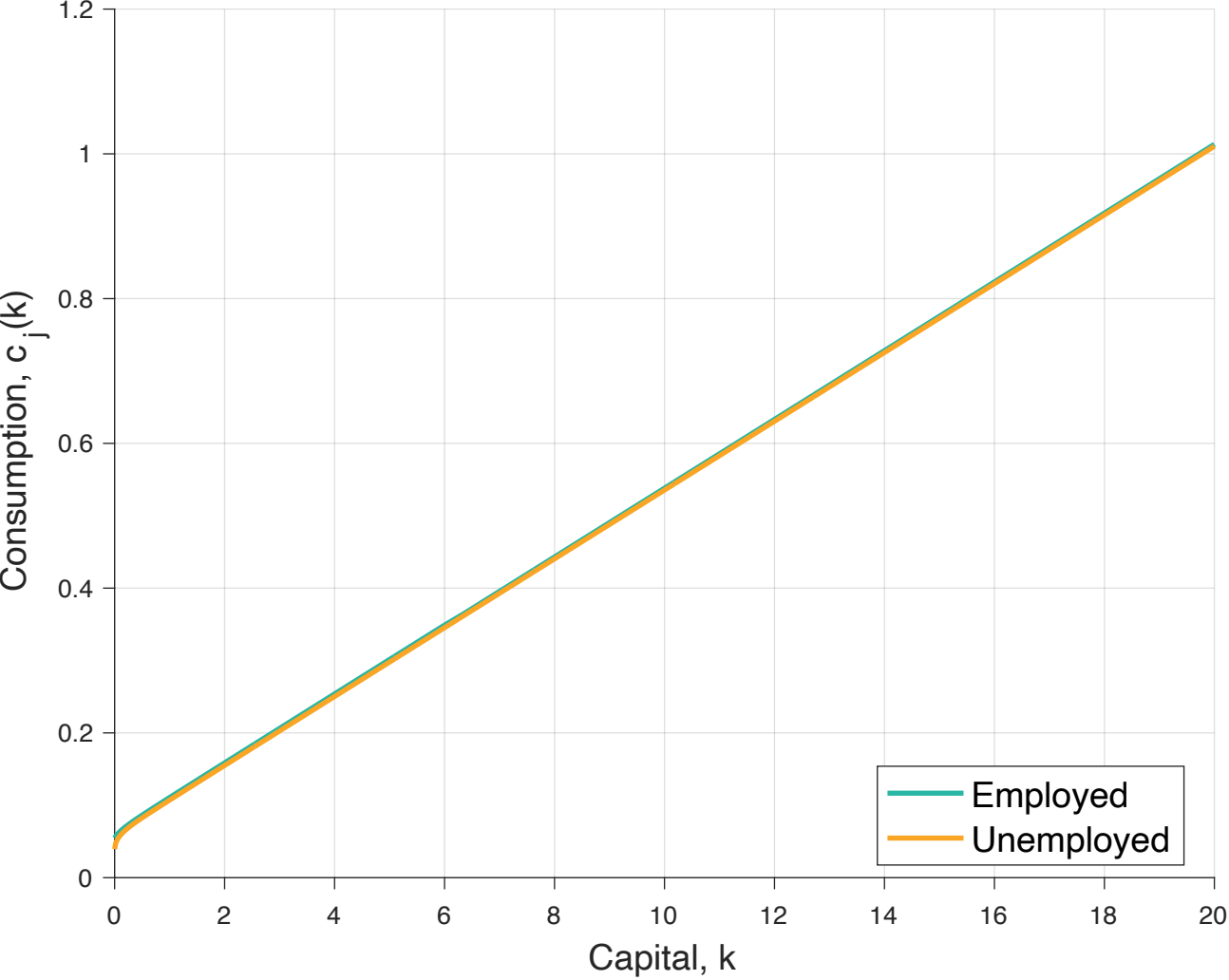
Question 2

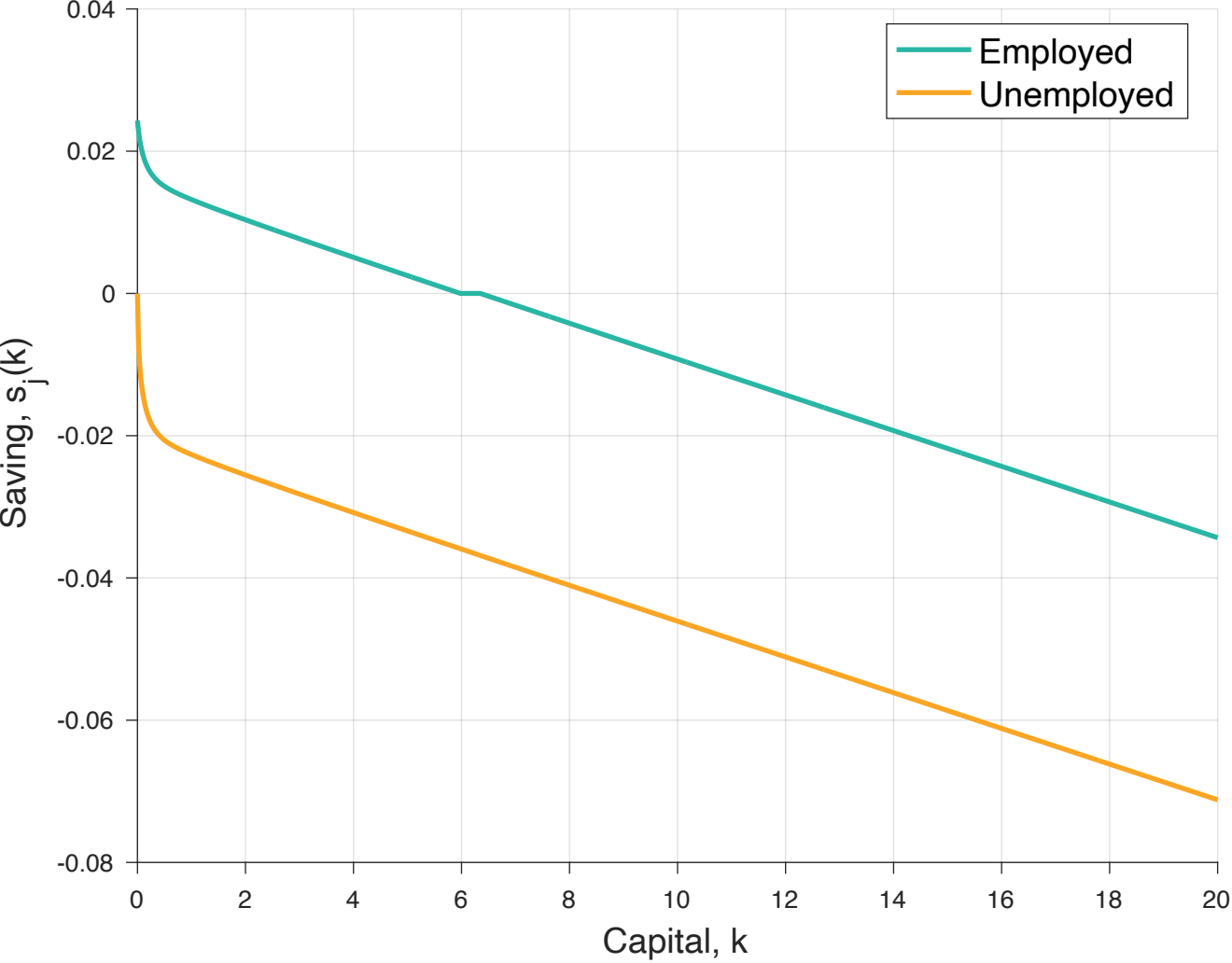

Employed
Unemployed
Densities, $g_j(k)$
Capital, k
0
0.5
1
1.5
2
2.5
3
3.5
4
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

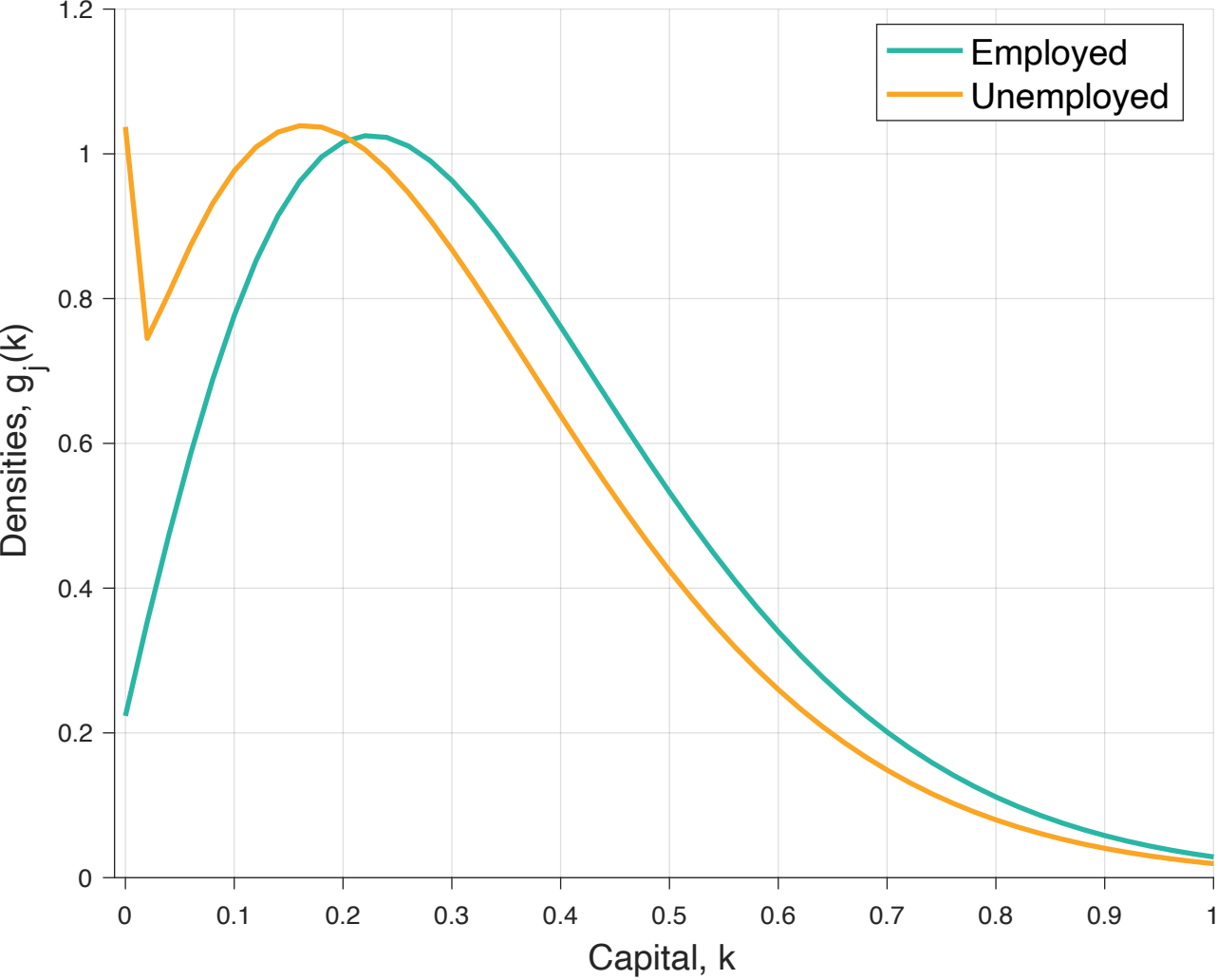
Question 3


Employed
Unemployed
Densities, $g_j(k)$
Capital, k
0
0.5
1
1.5
2
2.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

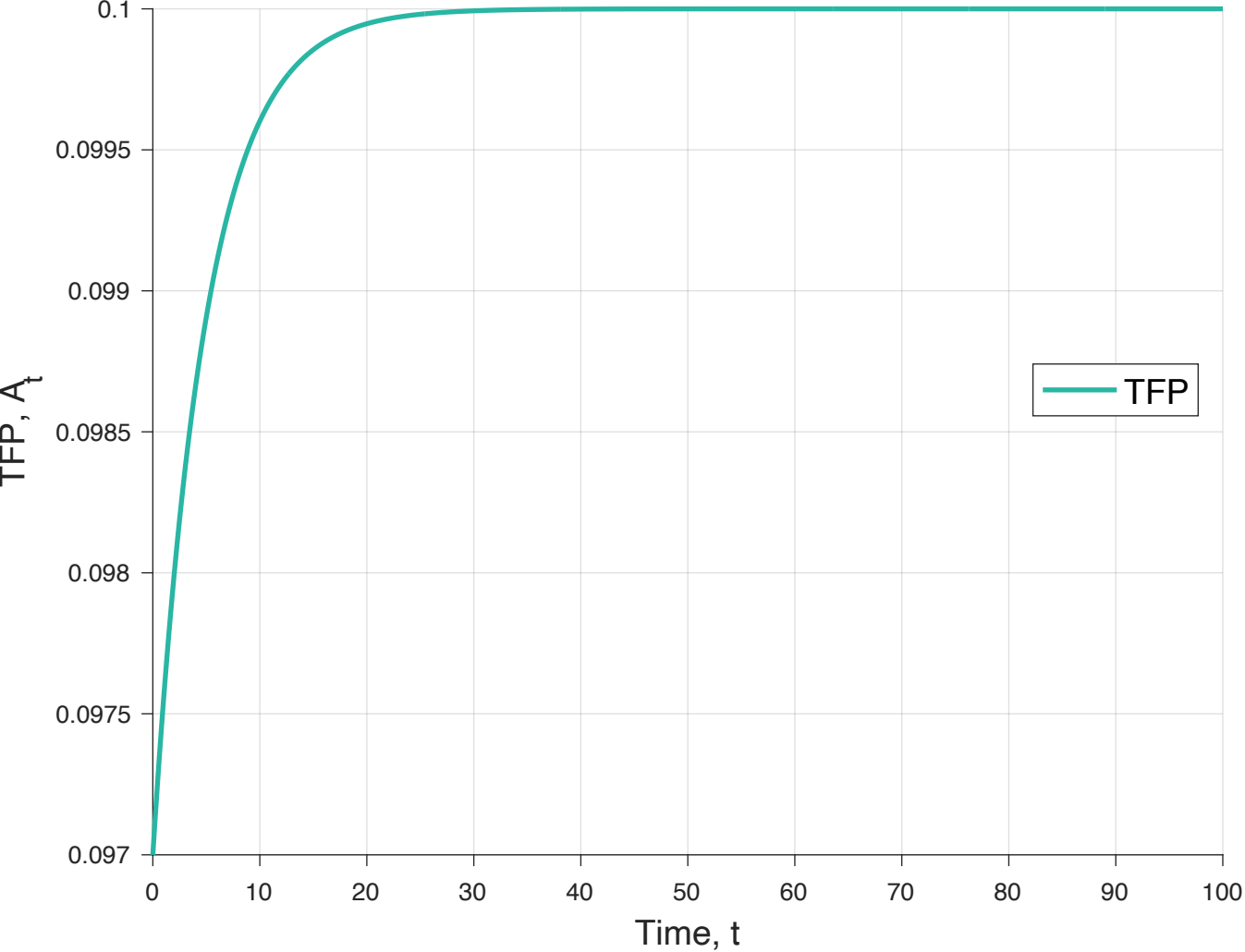
Question 6


1.2
1
0.8
0.6
0.4
0.2
0
Consumption, $c_j(k)$
0
2
4
6
8
10
12
14
16
18
20
Capital, k
Employed
Unemployed

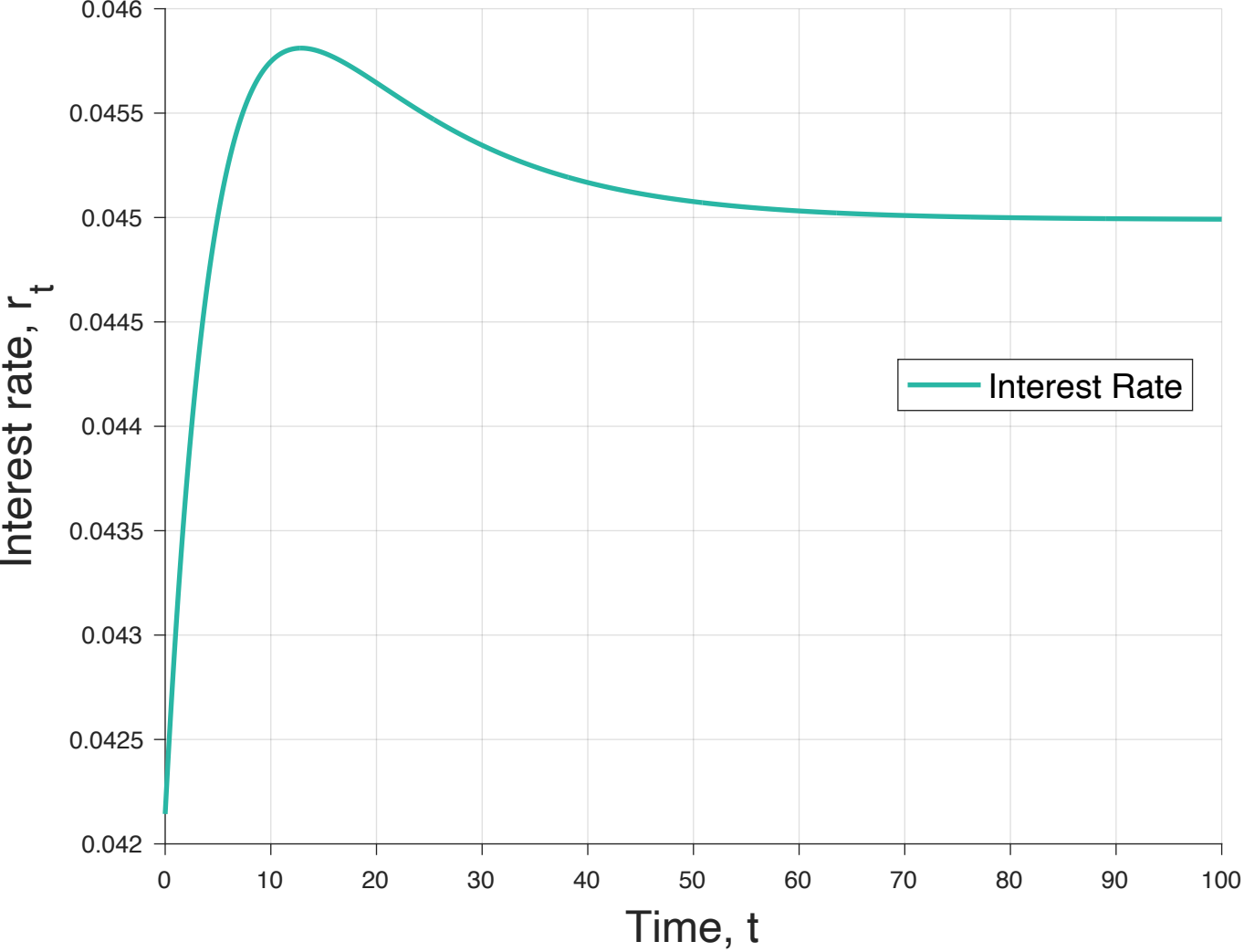

Employed
Unemployed
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
Saving, $s_j(k)$
0
2
4
6
8
10
12
14
16
18
20
Capital, k

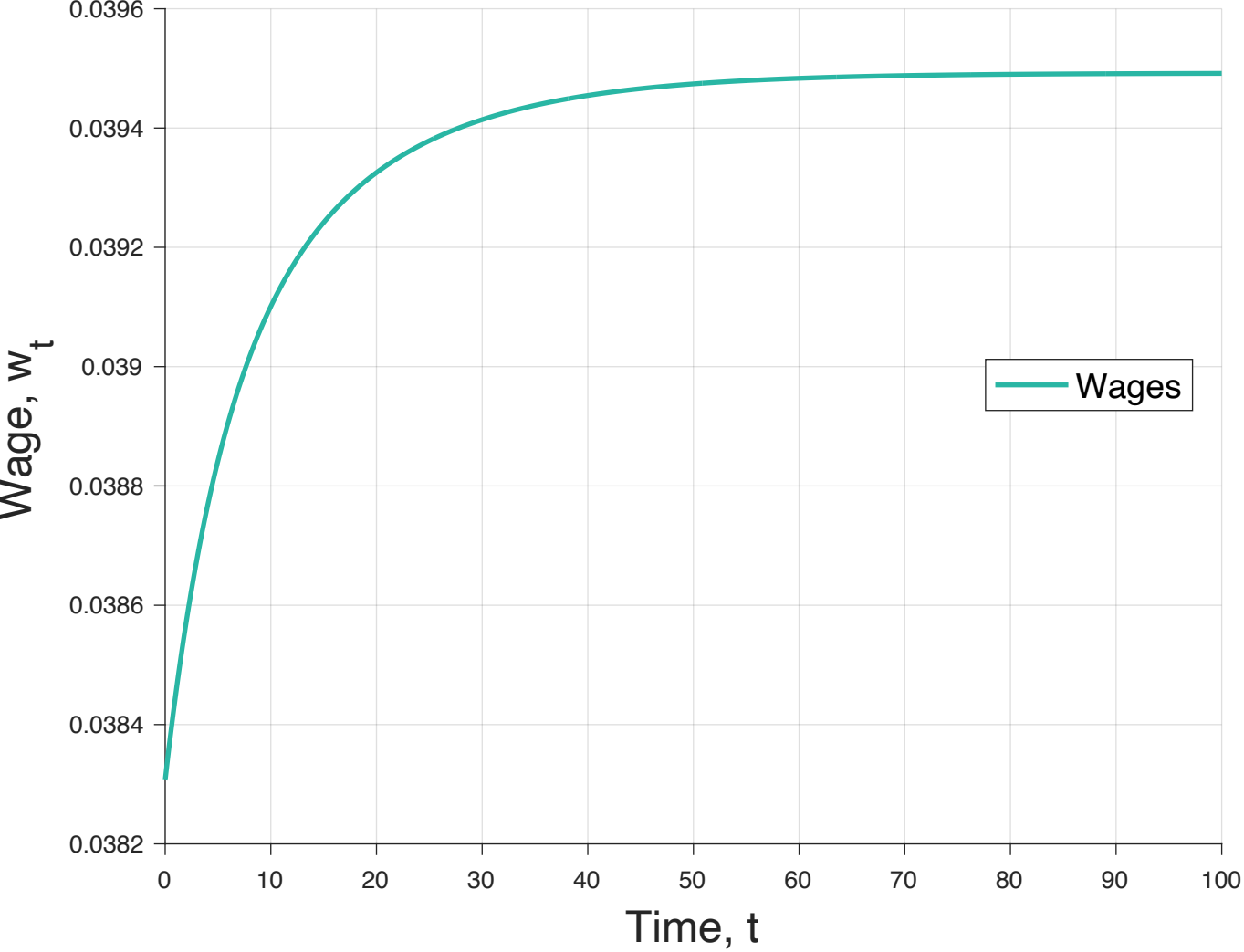

Employed
Unemployed
Densities, $g_j(k)$
Capital, k
0
0.2
0.4
0.6
0.8
1
1.2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

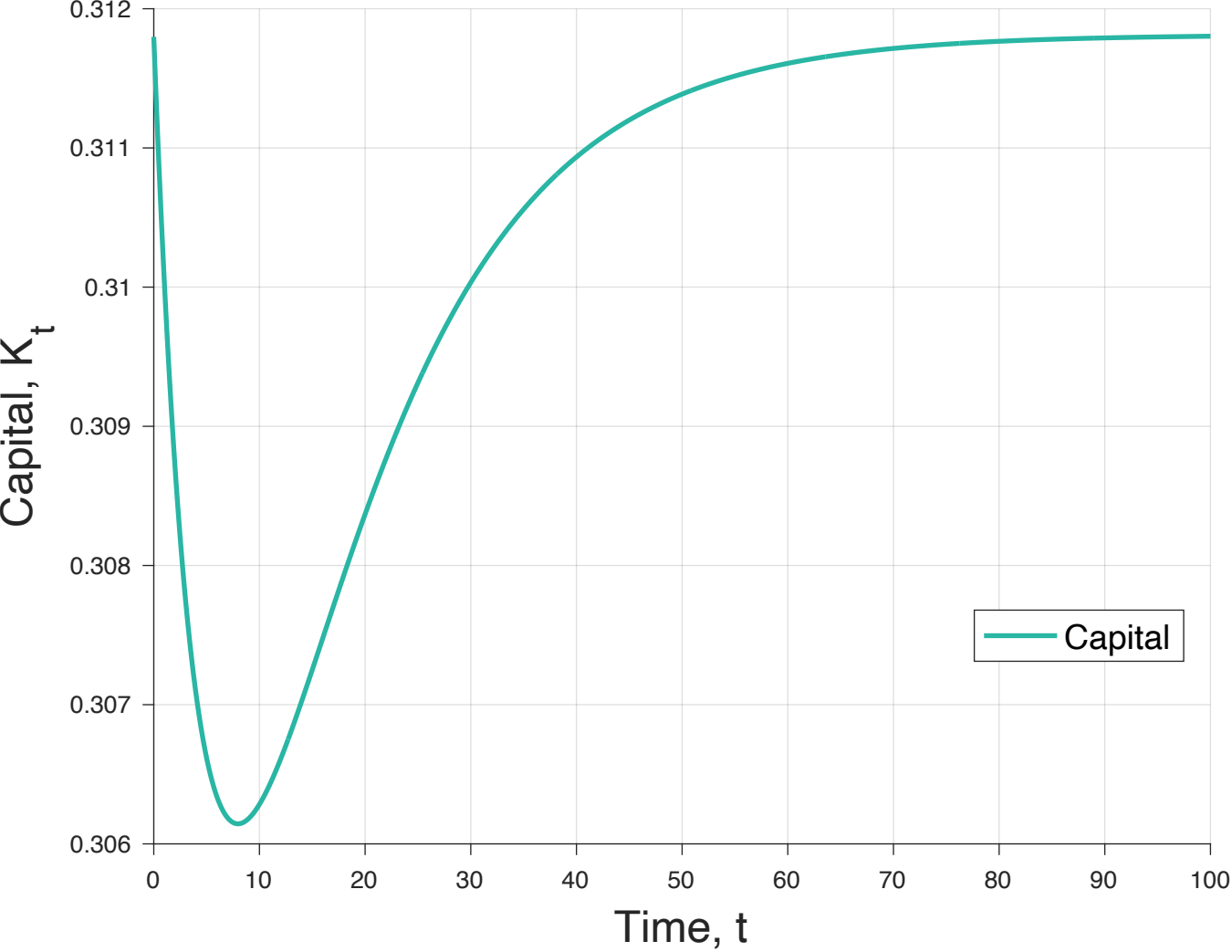
Question 7


0.1
0.0995
0.099
0.0985
0.098
0.0975
0.097
TFP, A_t
0
10
20
30
40
50
60
70
80
90
100
Time, t
TFP

Question 8


0.046
0.0455
0.045
0.0445
0.044
0.0435
0.043
0.0425
0.042
Interest rate, r_t
0
10
20
30
40
50
60
70
80
90
100
Time, t
Interest Rate


0.0396
0.0394
0.0392
0.039
0.0388
0.0386
0.0384
0.0382
Wage, w_t
0
10
20
30
40
50
60
70
80
90
100
Time, t
Wages


0.312
0.311
0.31
0.309
0.308
0.307
0.306
Capital, K_t
0
10
20
30
40
50
60
70
80
90
100
Time, t
Capital

problem_sets/problem_set_5/problem_set_5_question_1_2.m

```
%{
Purpose: Coding part of problem set 5
Created: Nico Rotundo 2024-12
%}

%% Define parameters and initialize grids

%{
Define parameters
%}

clear all;

% Relative risk aversion coefficient
sigma = 2;

% Productivity
z_g = 3;
z_b = 2;
z = [z_g, z_b];

% Transition rates
lambda_g = 1/3;
lambda_b = 1/2;
lambda = [lambda_g, lambda_b];

% Discount rate
rho = 0.05;

% Capital depreciation rate
delta = 0.05;

% Capital share in production
alpha = 1/3;

% TFP
A = 1;

% Interest rate for partial equilibrium part of the problem set
r = 0.035;

%{
Define capital grid
%}

% Borrowing constraint minimum
k_min = 0;

% Borrowing constraint maximum
k_max = 20;

% Number of grid points
num_points = 1000;

k_grid = linspace(k_min, k_max, num_points)';
dk = (k_max-k_min)/(num_points-1);

kk = [k_grid, k_grid]; % 1x2 matrix

%{
Define utility function and its derivative
%}

% Utility function (CRRA), handling vector inputs
U = @(c) (c.^(1 - sigma) / (1 - sigma));

% Derivative of utility, handling vector inputs
U_prime = @(c) c.^(-sigma);

% Inverse of derivative of utility
U_prime_inv = @(MU) (MU.^(-1/ sigma));

%{
Firm production technology and FOCs
%}

% Production technology Y = A K^alpha L^(1-alpha)
F = @(K,L) A.*K.^alpha.*L.^(1-alpha);

% F.C.S.
F_K = @(K,L) (1-alpha)*A*(L./K).^alpha;
F_fw = @(K,L) alpha*A*(K./L).^(1-alpha);
K_partial = @(L,r) (r(1*alpha)/(r+delta).*L).^(1/(1-alpha)) ;

%{
Tuning parameters (general)
%}

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

% The maximum number of value function iterations
max_iterations_vf = 100;

% Tolerance for value function iterations
tolerance = 10^(-6);

%{
Tuning parameters (interest rate iteration)
%}

% The maximum number of interest rate iterations
num_iterations_r = 1000;

% Tolerance for interest rate iterations
tolerance_3 = 10^(-5);

%% Initialize matrices before iterating

%{
Define differential operator
%}

% Forward: satisfies DFv=dvF
DF = sparse(num_points,num_points);
for i = 1:num_points
    DF(i,i) = -1/dk; DF(i,i+1) = 1/dk;
end
DF = sparse(DF);

% Backward: satisfies DBv=dvB
DB = sparse(num_points,num_points);
for i = 2:num_points
    DB(i,i-1) = -1/dk; DB(i,i) = 1/dk;
end
DB = sparse(DB);

%{
Define A-switch matrix
%}

A_switch = [speye(num_points),speye(num_points),speye(num_points),speye(num_points)];
            speye(num_points),speye(num_points),speye(num_points),speye(num_points)];


%{
Calculate labor, capital, and wage levels
%}

% Labor
L = F(z_ones(num_n,1)'lambda_ones(num_n,1));

% Capital
K = K_partial(L,r);

% Wage
w = w_fn(K,L);

%{
Define initial guesses
%}

% Guess for initial value of the interest rate
r_B_guess = 0.03;

% Set bounds on interest rate
r_low = 0.01;
r_high = 0.04;

% Define the number of points x 2 matrix
z = ones(num_points, 1);

% Initial guess for value function
V_B = U(w.*z + r.*kk) / rho;
V = V_B;

%% Value function iteration

% Loop over number of iterations for the value function
for n1=1:max_iterations_vf

    % Derivative of the forward value function
    dVf = DFv;

    % Derivative of the backward value function
    dVb = DBv;

    % Boundary condition on backwards value function (i.e., the borrowing constraint; a==a_min
    dVb(1,:) = U_prime(w.*z(1,:) + r.*kk(1,:));

    % Boundary condition on forward value function: a==a_max
    dVf(end,:) = U_prime(w.*z(end,:) + r.*kk(end,:));

    % Indicator whether value function is concave; for stability purposes
    I_concave = dVb > dVf;

    % Compute optimal consumption using forward derivative
    cf = U_prime_inv(dVf);

    % Compute optimal consumption using backward derivative
    cb = U_prime_inv(dVb);

    % Compute optimal savings using forward derivative
    sf = w.*z + r.*kk - cf;

    % Compute optimal savings using backward derivative
    sb = w.*z + r.*kk - cb;

    % Upwind scheme
    If = sf>0;
    Ib = sb<0;
    I0 = 1-If-Ib;
    dV0 = U_prime(w.*z + r.*kk);% If sf==0=sb, set s=0

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

    c = U_prime_inv(dV_upwind);

    % Update value function
    V_stacked = V(:);

    % Update consumption function
    c_stacked = c(:);

    u = U
    SD_a = spdiags([If(:,1).*sf(:,1), 0, num_points, num_points*4]+ spdiags([Ib(:,1).*sb(:,1), 0, num_points, num_points*4]0;
    SD_e = spdiags([If(:,2).*sf(:,2), 0, num_points, num_points*4]+ spdiags([Ib(:,2).*sb(:,2), 0, num_points, num_points*4]0;
    SD = [SD_a, sparse(num_points, num_points);
          sparse(num_points, num_points), SD_e];

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

    % B = (1/Rho + 1/Delta)I - P
    B = (rho + 1/Delta)*speye(2*num_points) - P;

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

    % V = B\b
    V_update = B\b;
    V_change = V_update - V_stacked;
    V = reshape(V_update, num_points, 2);

    % Convergence criterion
    disttol = max(abs(V_change));

    if disttol<tolerance
        disp("Value function converged, Iteration = ")
        disp(n1)
    break

    end
end

%% KF Equation

% Solve for G=gsto=P'*g
PT = P';
PT_rigs = PT;
gdot_stacked = zeros(2*num_points,1);

% Fix one value to obtain a non-singular matrix, otherwise matrix is singular
i_fix = 1;
gdot_stacked(i_fix)=1;

row_fix = [zeros(1,i_fix-1),1,zeros(1,2*num_points-i_fix)];
PT(i_fix,:) = row_fix;

g_stacked = PT\gdot_stacked;

% Normalization
g_sum = g_stacked*ones(2*num_points,1)*dk;
g_stacked = g_stacked./g_sum;

% Reshape
gg = reshape(g_stacked, num_points, 2);

% Solve KF equation
lg_stacked_eigs, eigvalf = eigs(PT_rigs, 1, 0);
g_sum_eigs = g_stacked_eigs'*ones(2*num_points,1)*dk;
g_stacked_eigs = g_stacked_eigs./g_sum_eigs;

%% Plots

%{
Question 1: Optimal consumption
%}

% Initialize plot
setfigs, 'FontSize', 18)

    % Employed
    plot(k_grid, c(:,2), 'LineWidth', 2, 'LineStyle', '-', 'Color', [41/255, 182/255, 164/255])
    hold on

    % Unemployed
    plot(k_grid, c(:,1), 'LineWidth', 2, 'LineStyle', '-', 'Color', [250/255, 165/255, 35/255])

    hold off

    grid
        xlabel('Capital, k','FontSize', 14)

        setgca, 'TickDir', 'out');
            box off;

        ylabel('Consumption, c_j(k)','FontSize', 14)

        xlim([k_min k_max])

        legend(sprintf('Employed', ...
            sprintf('Unemployed', ...
            sprintf('r=%.4f', r), 'Location', 'best', 'FontSize', 14)

% Export graph
exportgraphics(gcf, '/Users/nicorotundo/Documents/GitHub/DynamicProgramming2024/problem_sets/problem_set_5/output/question_1_consumption.pdf', 'ContentType', 'vector');

%% Question 1: Optimal savings

% Initialize plot
setfigs, 'FontSize', 18)

    % Employed
    plot(k_grid, kdot(:,2), 'LineWidth', 2, 'LineStyle', '-', 'Color', [41/255, 182/255, 164/255])
    hold on

    % Unemployed
    plot(k_grid, kdot(:,1), 'LineWidth', 2, 'LineStyle', '-', 'Color', [250/255, 165/255, 35/255])

    hold off

    grid
        xlabel('Capital, k', 'FontSize', 14)

        ylabel('Saving, s_j(k)', 'FontSize', 14)

        setgca, 'TickDir', 'out');
            box off;

        xlim([k_min k_max])

        legend(sprintf('Employed', ...
            sprintf('Unemployed', ...
            sprintf('r=%.4f', r), 'Location', 'best', 'FontSize', 14)

% Export graph
exportgraphics(gcf, '/Users/nicorotundo/Documents/GitHub/DynamicProgramming2024/problem_sets/problem_set_5/output/question_1_savings.pdf', 'ContentType', 'vector');

%% Question 1: Value function

% Initialize plot
setfigs, 'FontSize', 18)

    % Employed
    plot(k_grid, V(:,2), 'LineWidth', 2, 'LineStyle', '-', 'Color', [41/255, 182/255, 164/255])
    hold on

    % Unemployed
    plot(k_grid, V(:,1), 'LineWidth', 2, 'LineStyle', '-', 'Color', [250/255, 165/255, 35/255])

    hold off

    grid
        xlabel('Capital, k', 'FontSize', 14)

        ylabel('Value function, V_j(k)', 'FontSize', 14)

        setgca, 'TickDir', 'out');
            box off;

        xlim([k_min k_max])

        legend(sprintf('Employed', ...
            sprintf('Unemployed', ...
            sprintf('r=%.4f', r), 'Location', 'best', 'FontSize', 14)

% Export graph
exportgraphics(gcf, '/Users/nicorotundo/Documents/GitHub/DynamicProgramming2024/problem_sets/problem_set_5/output/question_1_value_function.pdf', 'ContentType', 'vector');

%% Question 2: Stationary distribution

%{
% Initialize plot
setfigs, 'FontSize', 18)

    % Employed
    plot(k_grid, gg(:,2), 'LineWidth', 2, 'LineStyle', '-', 'Color', [41/255, 182/255, 164/255])
    hold on

    % Unemployed
    plot(k_grid, gg(:,1), 'LineWidth', 2, 'LineStyle', '-', 'Color', [250/255, 165/255, 35/255])

    hold off

    grid
        xlabel('Capital, k', 'FontSize', 14)

        ylabel('Densities, g_j(k)', 'FontSize', 14)

        setgca, 'TickDir', 'out');
            box off;

        xlim([-0.05 1])

        legend(sprintf('Employed', ...
            sprintf('Unemployed', ...
            sprintf('r=%.4f', r), 'Location', 'best', 'FontSize', 14)

% Export graph
exportgraphics(gcf, '/Users/nicorotundo/Documents/GitHub/DynamicProgramming2024/problem_sets/problem_set_5/output/question_2.pdf', 'ContentType', 'vector');
```

problem_sets/problem_set_5/problem_set_5_question_3.m

```
%{
--------------------------------------------------------------------------------
Purpose: Coding part of problem set 5
Created: Nico Rotundo 2024-12
--------------------------------------------------------------------------------
%}

%% Define parameters and initialize grids

%{
--------------------------------------------------------------------------------
Define parameters
--------------------------------------------------------------------------------
%}

clear all;

% Relative risk aversion coefficient
sigma = 2;

% Productivity
z_u = 1;
z_e = 2;
z = [z_u, z_e];

% Transition rates
lambda_u = 1/3;
lambda_e = 1/3;
lambda = [lambda_u, lambda_e];

% Discount rate
rho = 0.05;

% Capital depreciation rate
delta = 0.05;

% Capital share in production
alpha = 1/3;

% TFP
A = .1;

% Interest rate for partial equilibrium part of the problem set
r = 0.04;

%{
--------------------------------------------------------------------------------
Define capital grid
--------------------------------------------------------------------------------
%}

% Borrowing constraint minimum
k_min = 0;

% Borrowing constraint maximum
k_max = 20;

% Number of grid points
num_points = 1000;

k_grid = linspace(k_min, k_max, num_points)';
dk = (k_max-k_min)/(num_points-1);

kk = [k_grid, k_grid]; % I*2 matrix

%{
--------------------------------------------------------------------------------
Define utility function and its derivative
--------------------------------------------------------------------------------
%}

% Utility function (CRRA), handling vector inputs
U = @(c) (c.^(1 - sigma)) / (1 - sigma);

% Derivative of utility, handling vector inputs
U_prime = @(c) c.^(-sigma);

% Inverse of derivative of utility
U_prime_inv = @(Vp) (Vp).^(-1 / sigma);

%{
--------------------------------------------------------------------------------
Firm production technology and FOCs
--------------------------------------------------------------------------------
%}

% Production technology Y = A K^alpha L^(1-alpha)
Y = @(K,L) K.^alpha*L.^(1-alpha);

% F.O.C.s:
w_foc = @(K,L) (1-alpha)*A*(K/L).^alpha;
r_foc = @(K,L) alpha*A*(L/K).^(1-alpha);
K_partial = @(L,r) L*((alpha*A/(r+delta)).^(1/(1-alpha))) ;

%{
--------------------------------------------------------------------------------
Tuning parameters (general)
--------------------------------------------------------------------------------
%}

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

% The maximum number of value function iterations
max_iterations_vf = 100;

% Tolerance for value function iterations
tolerance = 10^(-6);

 %{
--------------------------------------------------------------------------------
Tuning parameters (interest rate iteration)
--------------------------------------------------------------------------------
%}

% The maximum number of interest rate iterations
num_iterations_r = 1000;

% Tolerance for interest rate iterations
tolerance_S = 10^(-5);

%% Initialize matrices before iterating

%{
--------------------------------------------------------------------------------
Define differential operator
--------------------------------------------------------------------------------
%}

% Forward; satisfies DfxV=dvDf
Df = zeros(num_points, num_points);
for i = 1:num_points-1
    Df(i,i) = -1/dk; Df(i,i+1) = 1/dk;
end
Df = sparse(Df);

% Backward; satisfies DbxV=dV
Db = zeros(num_points, num_points);
for i = 2:num_points
    Db(i,i-1) = -1/dk; Db(i,i) = 1/dk;
end
Db = sparse(Db);

%{
--------------------------------------------------------------------------------
Define A-switch matrix
--------------------------------------------------------------------------------
%}

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


%{
--------------------------------------------------------------------------------
Calculate labor, capital, and wage levels
--------------------------------------------------------------------------------
%}

% Labor
L = [z_e*lambda_u ; z_u*lambda_e]/(lambda_e+lambda_u);

% Capital
K = K_partial(L,r);

% Wage
w = w_foc(K,L);

%{
--------------------------------------------------------------------------------
Define initial guesses
--------------------------------------------------------------------------------
%}

% Guess for initial value of the interest rate
r_0_guess = 0.03;

% Set bounds on interest rate
r_min = 0.01;
r_max = 0.04;

% Define the number of points * 2 matrix
z = ones(num_points, 1).*z;

% Initial guess for value function
V_0 = U(w.*z + r.*kk)./rho;
V = V_0;

%% Value function iteration

% Loop over number of interations for the value function
for n=1:max_iterations_vf

    % Derivative of the forward value function
    dVf = Df*V;

    % Derivative of the backward value function
    dVb = Db*V;

    % Boundary condition on backwards value function i.e., the borrowing constraint; a==a_min
    dVb(1,:) = U_prime(w.*z(1,:) + r.*kk(1,:));

    % Boundary condition on forward value function; a==a_max
    dVf(end,:) = U_prime(w.*z(end,:) + r.*kk(end,:));

    % Indicator whether value function is concave; for stability purposes
    I_concave = dVb > dVf;

    % Compute optimal consumption using forward derivative
    cf = U_prime_inv(dVf);

    % Compute optimal consumption using backward derivative
    cb = U_prime_inv(dVb);

    % Compute optimal savings using forward derivative
    sf = w.*z + r.*kk - cf;

    % Compute optimal savings using backward derivative
    sb = w.*z + r.*kk - cb;

    % Upwind scheme
    If = sf>0;
    Ib = sb<0;
    I0 = 1-If-Ib;
    dV0 = U_prime(w.*z + r.*kk); % If sf<=0<=sb, set s=0

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

    c = U_prime_inv(dV_upwind);

    % Update value function
    V_stacked = V(:);

    % Update consumption function
    c_stacked = c(:);

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

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

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

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

    % V = B\b;
    V_update = B\b;
    V_change = V_update - V_stacked;
    V = reshape(V_update, num_points, 2);

    % Convergence criterion
    dist(n) = max(abs(V_change));

    if dist(n)<tolerance
        disp('Value function converged. Iteration = ')
        disp(n)
    break

    end

end

%% KF Equation

% Solve for 0=gdot+P'*g
PT = P';
PT_eigs = PT;
gdot_stacked = zeros(2*num_points,1);

% Fix one value to obtain a non-singular matrix, otherwise matrix is singular
i_fix = 1;
gdot_stacked(i_fix)=.1;

row_fix = [zeros(1,i_fix-1),1,zeros(1,2*num_points-i_fix)];
PT(i_fix,:) = row_fix;

g_stacked = PT\gdot_stacked;

% Normalization
g_sum = g_stacked'*ones(2*num_points,1)*dk;
g_stacked = g_stacked./g_sum;

% Reshape
gg = reshape(g_stacked, num_points, 2);

% Solve KF equation
[g_stacked_eigs, eigval] = eigs(PT_eigs, 1, 0);
g_sum_eigs = g_stacked_eigs'*ones(2*num_points,1)*dk;
g_stacked_eigs = g_stacked_eigs./g_sum_eigs;

%% Plots

%{
--------------------------------------------------------------------------------
Question 3: Stationary distribution at r = 0.04
--------------------------------------------------------------------------------
%}

% Initialize plot
setfigca, 'FontSize', 18)

    % Employed
    plot(k_grid, gg(:,2), 'LineWidth', 2, 'LineStyle', '-', 'Color', [41/255, 182/255, 164/255])
    hold on

    % Unemployed
    plot(k_grid, gg(:,1), 'LineWidth', 2, 'LineStyle', '-', 'Color', [250/255, 165/255, 35/255])

    hold off

    grid
        xlabel('Capital, k', 'FontSize', 14)

        ylabel('Densities, g_j(k)', 'FontSize', 14)

        set(gca, 'TickDir', 'out');
            box off;

        xlim([-.01 11])

        legend(sprintf('Employed'), ...
            sprintf('Unemployed'), ...
            sprintf('r=%.4f', r), 'Location', 'best', 'FontSize', 14)

% Export graph
exportgraphics(gcf, "/Users/nicorotundo/Documents/GitHub/DynamicProgramming2024/problem_sets/problem_set_5/output/question_3.pdf", 'ContentType', 'vector');
```

portfolio_optimization_and_hedging_under_transaction_costs:_a_stochastic_control_approach

```
# -*- coe -*-
# -*- mode: org -*-
# Persons: Yuheng Wei and Andrew Lim
# Time-stamp: Sun Jan 19 23:59:59 2024
# Time-stamp: Sun Jan 19 23:59:59 2024

# ==========================================================================
# ==========================================================================
# ==========================================================================
```