Coding Exercise 2 Solution: Solving Steady States and Dynamic Paths in the Ramsey-Cass-Koopmans Model

ECON 202A

November 5, 2024

1. Analytical Steady States

The steady-state values for capital k^* and consumption c^* are given by:

$$k^* = \left(\frac{\rho + \theta g}{\alpha A}\right)^{\frac{1}{\alpha - 1}}$$

$$c^* = A(k^*)^\alpha - (n+g)k^*$$

Calculating these with the given parameters:

$$k^* = \left(\frac{0.03 + 1 \cdot 0.02}{\frac{1}{3} \cdot 1}\right)^{\frac{1}{\frac{1}{3} - 1}} \approx 17.2133$$

$$c^* = 1 \cdot (17.2133)^{\frac{1}{3}} - (0.02 + 0.02) \cdot 17.2133 \approx 1.8935$$

2. Numerical Steady States Using `fsolve`

Using `fsolve` in MATLAB, we define the residual function for k:

$$f_k(k) = f'(k) - \rho - \theta g$$

and solve it numerically to find k^*. Substituting k^* back into the production function, we get:

$$c^* = A(k^*)^\alpha - (n+g)k^*$$

The MATLAB solution yields $k^* \approx 17.1414$ and $c^* \approx 1.8927$, closely matching the analytical values.

3. Numerical Steady States Using Newton's Method

Using Newton's method in MATLAB without `fsolve`, we iteratively solve the steady-state residual function:

$$f_k = f'(k) - \rho - \theta g$$

and update k at each iteration as:

$$k_{\text{new}} = k - \frac{f_k}{f'_k}$$

With Newton's method, we find $k^* \approx 17.2133$ and $c^* \approx 1.8935$, verifying the analytical solution.

4. Comparison of Analytical and Numerical Solutions

The analytical approach, `fsolve`, and Newton's method all yield consistent results for $k^* \approx 17.2$ and $c^* \approx 1.89$, confirming the accuracy of the numerical methods.

5. Simulating Dynamic Paths Using the Shooting Algorithm

The shooting algorithm is implemented in MATLAB. Starting with an initial guess for consumption c_0, we iteratively adjust c_0 to satisfy the transversality condition at the terminal time T. The paths for $k(t)$, $c(t)$, and the implied rate of return $r(t) = f'(k)$ are plotted over time.

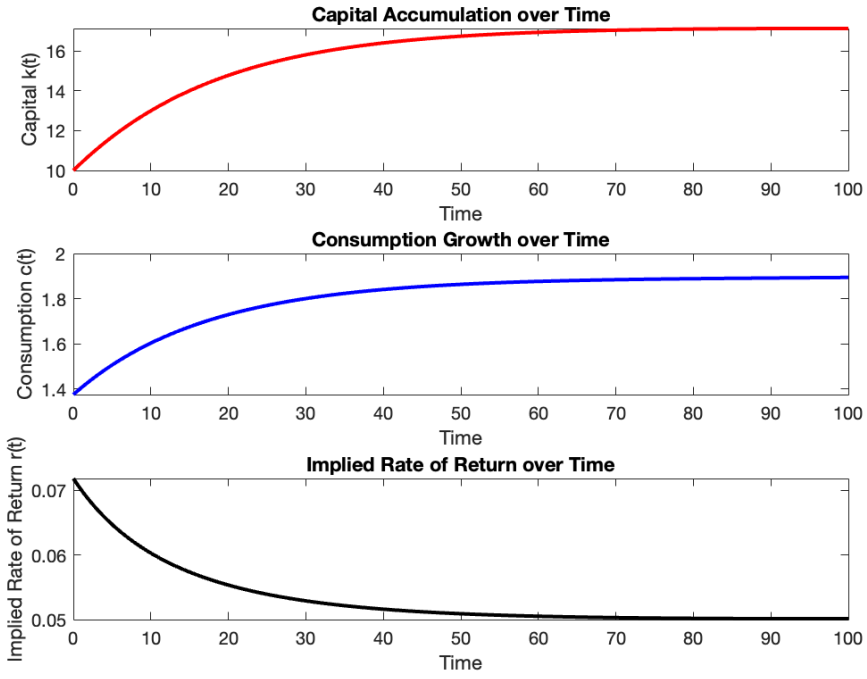

Figure 1: Paths of Capital, Consumption, and Implied Rate of Return Over Time

As shown in Figure 1, capital, consumption, and the implied rate of return converge towards their steady-state values over time.

6. Calculating Implied Market-Clearing Interest Rate $r_{\text{market}}(t)$

To calculate the implied market-clearing interest rate under the assumption of a competitive capital market, we use the following relation:

$$k_d(r_{\text{market}}(t)) - k_s = \left(\frac{r(t)}{\alpha A}\right)^{\frac{1}{\alpha-1}} - k(t)$$

where $k(t)$ is the capital path obtained from the shooting algorithm. Here, r_{market} represents the interest rate at which capital demand K_d equals capital supply K_s.

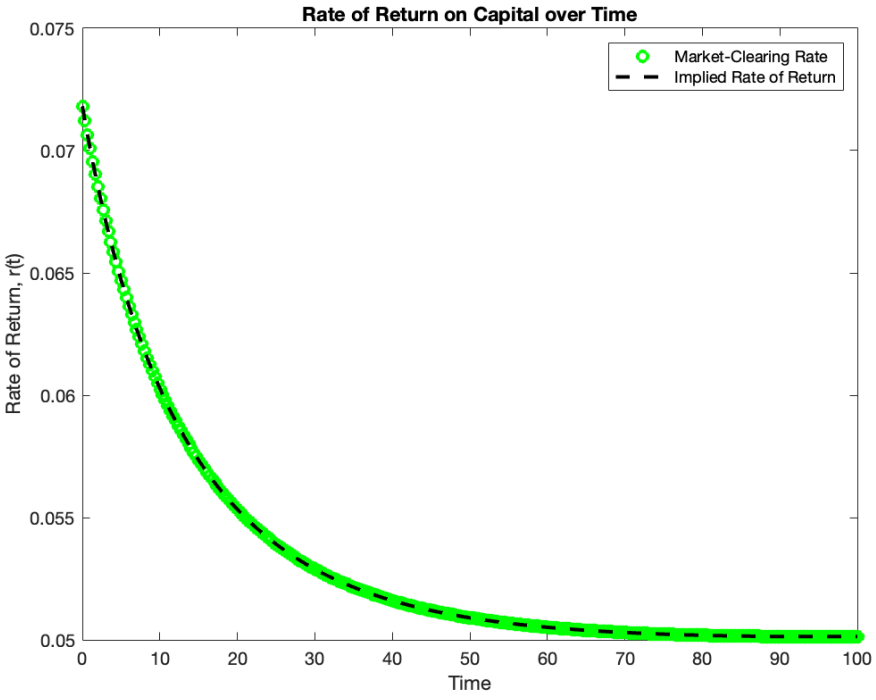

Figure 2: Market-Clearing Rate and Implied Rate of Return Over Time

Using MATLAB's `fsolve` function, we find that r_{market} matches the implied rate of return $r(t)$ over time, verifying market equilibrium.

7. Interpretation of Results

These results indicate that the market-clearing interest rate in a competitive equilibrium would match the implied rate of return if the decentralized market achieved the planner's optimal path, thus linking the planner's solution to the competitive equilibrium.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MATLAB Script: ramsey_steady.m
%
% Author: Kiyea Jin
% Date: October 28, 2024
%
% Description:
% This script analytically and numerically solve for the steady states
% and simulate dynamic paths in the continuous-time Ramsey-Cass-Koopmans
% model.
%
% The model consists of two equations:
%   (dc/dt)/c = (f'(k) - rho - theta*g)/theta
%   dk/dt = f(k) - c - (n+g)k
% where an initial condition for capital K0 is provided,
% and a transversality conidtion is imposed as a terminal condition.
%
% Parameters:
% - Discount rate (rho): 0.03
% - Inverse of IES (theta): 1
% - Technology growth rate (g): 0.02
% - Population growth rate (n): 0.02
% - Capital share (alpha): 1/3
% - TFP (A): 1
% - Initial boundary condition: K0 = 10
%
% Code Structure:
% 1. DEFINE PARAMETERS
% 2. INITIALIZE GRID POINTS
% 3. STEADY STATES
%   3-1. ANALYTICAL STEADY STATES
%   3-2. NUMERICAL STEADY STATES USING FSOLVE
%   3-3. NUMERICAL STEADY STATES USING NEWTON'S METHOD
% 4. SHOOTING ALGORITHM
% 5. PLOT
% 6. MARKET CLEARING INTEREST RATES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
close all;
clc;

%% 1. DEFINE PARAMETERS

p = define_parameters();

%% 2. INITIALIZE GRID POINTS

t = linspace(p.tmin, p.tmax, p.I)';
dt = (p.tmax-p.tmin)/(p.I-1);

%% 3-1. ANALYTICAL STEADY STATES
% kss = ((rho + theta*g)/(alpha*A))^(1/(alpha-1))
```

```
% css = Ak^alpha - (n+g)*kss

kss_analytical = ((p.rho + p.theta * p.g) / (p.alpha * p.A))^(1 / (p.alpha - 1));
css_analytical = p.f(kss_analytical) - (p.n+p.g) * kss_analytical;

%% 3-2. NUMERICAL STEADY STATES USING FSOLVE
% (dc/dt)/c = (f'(k) - rho - theta*g)/theta
% dk/dt = f(k) - c - (n+g)k

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

cdot = @(k) p.f_prime(k) - p.rho - p.theta*p.g;
kss_numerical = fsolve(cdot, p.k0, options);

css_numerical = p.f(kss_numerical) - (p.n+p.g) * kss_numerical;

fprintf('Numerical steady-state capital (K*): %.4f\n', kss_numerical);
fprintf('Numerical steady-state consumption (C*): %.4f\n\n', css_numerical);

%% 3-3. NUMERICAL STEADY STATES USING NEWTON'S METHOD
% (dc/dt)/c = (f'(k) - rho - theta*g)/theta
% dk/dt = f(k) - c - (n+g)k

k_guess = p.k0;

for n = 1:p.maxit

    % Residual function and derivative for capital steady state
    f_k = p.f_prime(k_guess) - p.rho - p.theta*p.g;
    f_k_prime = (p.alpha*(p.alpha-1)*p.A*k_guess^(p.alpha-2));

    % Update using Newton's method
    k_new = k_guess - f_k / f_k_prime;

    % Check convergence
    if abs(k_new - k_guess) < p.tol
        fprintf('Numerical steady-state capital (K*) converged to %.4f after %d iterat
        break;
    end

    k_guess = k_new;
end

kss_numerical_2 = k_new;
css_numerical_2 = p.f(kss_numerical_2) - (p.n + p.g) * kss_numerical_2;

fprintf('Numerical steady-state capital (K*): %.4f\n', kss_numerical_2);
fprintf('Numerical steady-state consumption (C*): %.4f\n\n', css_numerical_2);

%% 4. SHOOTING ALGORITHM

    % 4-1. Guess an initial value of consumption

    c0_guess = 1;
```

```
    c0 = c0_guess;

    % 4-2. Find the initial value of consumption that satisfies transversality conditi

        % Objective function that calculates the difference between terminal capital k
        % Note: k(T)-kss is a function of c0
        diff = @(c0) terminal_condition(c0, p.k0, kss_numerical, p.f, p.f_prime, p.rho

        % Use fsolve to find the initial consumption c0 that makes k(T) = k_ss
        % Note: X = fsolve(FUN, X0, options) starts at the matrix X0 and tries to solv
        % Set OPTIONS = optimoptions('fsolve','Algorithm','trust-region'), and then pa
        options = optimoptions('fsolve', 'TolFun', p.tol, 'Display', 'iter');
        c0 = fsolve(diff, c0, options);

    % 4-3. Forward simulate with the updated initial consumption

    [k, c] = forward_simulate(c0, p.k0, p.f, p.f_prime, p.rho, p.theta, p.g, p.n, dt,

    % 4-4. Calculate implied rate of return of capital
    r = p.f_prime(k);

%% 5. PLOT

% Evolution of capital, consumption, and implied rate of return of capital

figure;
subplot(3,1,1);
plot(t, k, 'r-', 'LineWidth', 2);
xlabel('Time'); ylabel('Capital k(t)');
title('Capital Accumulation over Time');

subplot(3,1,2);
plot(t, c, 'b-', 'LineWidth', 2);
xlabel('Time'); ylabel('Consumption c(t)');
title('Consumption Growth over Time');

subplot(3,1,3);
plot(t, r, 'k-', 'LineWidth', 2);
xlabel('Time'); ylabel('Implied Rate of Return r(t)');
title('Implied Rate of Return over Time');

%% 6. MARKET CLEARING INTEREST RATES

r0 = (p.rho+p.theta*p.g)*ones(p.I,1);

market_clearing = @(r) (r./(p.alpha*p.A)).^(1/(p.alpha-1)) - k;
r_market = fsolve(market_clearing, r0, options);

% Plot the market-clearing rate and the implied rate of return
figure;
plot(t, r_market, 'go', 'LineWidth', 2);
hold on;
plot(t, r, 'k--', 'LineWidth', 2);
xlabel('Time');
```

```
ylabel('Rate of Return, r(t)');
legend('Market-Clearing Rate', 'Implied Rate of Return');
title('Rate of Return on Capital over Time');
hold off;
```

```
function p = define_parameters()

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

%% Economic Parameters

    % Discount rate
    p.rho = 0.03;

    % Inverse of IES
    p.theta = 1;

    % Technology growth rate
    p.g = 0.02;

    % Population growth rate
    p.n = 0.02;

    % Capital share
    p.alpha = 1/3;

    % TFP
    p.A = 1;

%% Economic Functions

    % Production function
    p.f = @(k) p.A * k.^p.alpha;

    % MPK
    p.f_prime = @(k) p.alpha * p.A * k.^(p.alpha - 1);

%% Boundary Conditions

    % Initial capital
    p.k0 = 10;

%% Grid Paramters

    p.tmin = 0;
    p.tmax = 100;

    % The number of time steps
    p.I = 300;

%% Newton Method Tuning Parameters
```

```
    % Tolerance for Newton's method
    p.tol = 1e-6;

    % Maximum iterations for Newton's method
    p.maxit = 1000;

end
```