1 Two-period OLG

Premise:

1. Agents are homogenous in preferences.
2. Each agent lives for 2 periods: young and old. They work inelastically when young and retire when old.
3. Agents are perfect foresight.

1.1 Simple Diamond's Model

Household

Preferences

$$U(c_t, d_{t+1}) = u(c_t) + \beta u(d_{t+1}), \tag{1}$$

subject to constraints

$$\begin{aligned} c_t + s_t &= w_t, \\ d_{t+1} &= R_{t+1} s_t. \end{aligned}$$

The functional form of utility can take one of the following [1]

$$u(c) = \begin{cases} \dfrac{\sigma}{\sigma - 1}(c^{1-\frac{1}{\sigma}} - 1) & \text{if } \sigma > 0, \sigma \neq 1 \text{ (CIES)}, \\ \dfrac{c^{1-\sigma} - 1}{1 - \sigma} & \text{if } \sigma \geq 0, \sigma \neq 1 \text{ (CRRA)}, \\ \ln(c) & \text{if } \sigma = 1. \end{cases}$$

The population grows with a deterministic rate of n.

1. Solve the agent's problem using log utility by maximizing (1) subject to the two budget constraints.

$$s_t = \frac{\beta}{1+\beta} w_t, \tag{2}$$

$$c_t = \frac{1}{1+\beta} w_t, \tag{3}$$

$$d_t = \frac{\beta}{1+\beta} R_{t+1} w_t. \tag{4}$$

2. Solve the agent's problem to get $s_t(w_t, R_{t+1})$ using CIES utility as follows

$$u(c) = \frac{\sigma}{\sigma - 1} c^{1-1/\sigma}.$$

   You should obtain

$$s_t = \frac{\beta}{1 + \beta^{-\sigma} R_{t+1}^{1-\sigma}} w_t. \tag{5}$$

[1] The constant term (-1) can be omitted in CIES and CRRA.

Production

A representative firm maximizes its profit with a Cobb-Douglas production technology

$$Y_t = F(K_t, L_t) = AK_t^{\alpha} L_t^{1-\alpha}.$$

where $\alpha \in (0,1), A > 0$ and profit

$$\Pi_t = Y_t - w_t L_t - R_t K_t.$$

Define the capital-labor ratio as

$$k_t = \frac{K_t}{L_t}.$$

Solve the firm problem to obtain $w_t(k_t)$ and $R_t(k_t)$.

$$w_t = (1-\alpha) A k_t^{\alpha}, \tag{6}$$
$$R_t = \alpha A k_t^{\alpha-1}. \tag{7}$$

Alternatively, we can use a more general production function of the CES form

$$F(K_t, L_t) = A[\alpha K_t^{-\rho} + (1-\alpha) L_t^{-\rho}]^{-1/\rho}.$$

with $\alpha \in (0,1), A > 0, \rho > -1, \rho \neq 0$. The elasticity of substitution between K and L is $1/(1+\rho)$. An increase in ρ implies the two inputs become less substitutable. In this case, we can obtain the temporal equilibrium.

$$w_t = A(1-\alpha)(\alpha k_t^{-\rho} + 1 - \alpha)^{-(1+\rho)/\rho},$$
$$R_t = A\alpha(\alpha k_t^{-\rho} + 1 - \alpha)^{-(1+\rho)/\rho}.$$

Intertemporal Equilibrium

Labor market clears

$$L_t = N_t.$$

Capital market clears

$$K_{t+1} = s_t N_t,$$

in capital-labor ratio

$$k_{t+1} = \frac{K_{t+1}}{L_{t+1}} = \frac{s_t}{1+n}. \tag{8}$$

Goods market clears

$$Y_t = N_{t-1} d_t + N_t (c_t + s_t).$$

Definition 1 (Intertemporal Equilibrium)**.** Given initial capital-labor ratio $k_0 > 0$, the intertemporal equilibrium is defined as a sequence of factor prices $\{R_t, w_t\}_{t=1}^{\infty}$, a sequence of allocations for young agents' consumptions and saving $(d_1, \{c_t, s_t, d_{t+1}\}_{t=1}^{\infty})$, and a sequence of firm allocation $\{K_t, L_t\}_{t=1}^{\infty}$ such that

1. Factor prices determined $\{R_t, w_t\}_{t=1}^{\infty}$ by Eqs.(6),(7) and solve the firm problem

2. The allocation $(d_1, \{c_t, s_t, d_{t+1}\}_{t=1}^{\infty})$ defined by Eqs.(2),(3),(4) solve the household problem.

3. All markets (labor, capital, goods) clear.

Show that the law of motion for the capital-labor ratio is

1. With Cobb-Douglas technology and log utility:

$$k_{t+1} = \phi(k_t) = \frac{\beta A(1-\alpha)}{(1+n)(1+\beta)} k_t^{\alpha}. \tag{9}$$

2. With CIES technology ($\rho < 0$) and log utility:

$$k_{t+1} = \frac{\beta A(1-\alpha)(\alpha k_t^{-\rho} + 1 - \alpha)^{-(1+\rho)/\rho}}{(1+n)(1+\beta)}. \tag{10}$$

Steady State

In the steady state, it must be that

$$k_{t+1} = k_t = k^*.$$

Using the above, solve Eq.(9) for an analytical solution

$$k^* = \left(\frac{\beta A(1-\alpha)}{(1+n)(1+\beta)} \right)^{\frac{1}{1-\alpha}}. \tag{11}$$

Obviously, solving Eq.(10) by hand is impossible. In such a case, we can use a computer algorithm to find the solution.

Parameters

Parameters	Value
β	0.99^{30}
α	0.3
ρ	-1.5
A	10
n	0.3

With this set of parameters, you can calculate k^* from Eq.(11) as 3.26519. Now, we are going to solve it numerically using information from Eq.(9) only.

1.2 Nonlinear Solver

There are many ways to solve Eq.(9) numerically for the steady state. First, we can prove that a solution exists.

Exercise 1. Use intermediate value theorem on Eq.(9) to prove the existence of k^*. (Adv). Do similarly for Eq.(10).

Then, we rewrite it to

$$k - \frac{\beta A(1-\alpha)}{(1+n)(1+\beta)} k^{\alpha} = 0. \tag{12}$$

Solving for k is the same as finding the root of this equation. We are going to see how some algorithms (such as Bisection, Newton, and Secant) work in finding this root. For illustrations, visit Appendix A.1.

Bisection

The function must be continuous on an interval $[a, b]$. Furthermore, $f(a)$ and $f(b)$ must have opposite signs. The algorithm helps us find $p \in [a, b]$ s.t. $f(p) = 0$. First, choose a point halfway between a and b and check its sign. It will have the same sign as either $f(a)$ or $f(b)$. If it has the same sign as a, assign c as the new a and repeat.

1. Choose a and b such that $f(a) \times f(b) < 0$.
2. Set $i = 1$.
3. Calculate $f(a)$.
4. Choose a point c where
$$c = a + \frac{b-a}{2}.$$
5. Calculate $f(c)$ and evaluate its sign.
   (a) if $f(c) = 0$ or $(b-a)/2 < tol$, end and output c.
   (b) if $f(c) \cdot f(a) > 0$
      i. assign: a = c,
      ii. $f(a) = f(c)$
   (c) Else, assign $b = c$.
6. Repeat step 2 by setting $i = i + 1$.

Newton-Raphson

Commonly used when you can obtain the derivative of the function nicely. This method approximates a function by its tangent line to get a successively better estimate of the roots. Let n be the n^{th} iteration, the general formula for the next value is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{13}$$

1. Provide an initial guess c_0.
2. Set $i = 1$.
3. Set $c = c_0 - \frac{f(c_0)}{f'(c_0)}$.
4. Calculate $\varepsilon = |c - c_0|$:
   (a) if $\varepsilon < tol$, end the program and output c.
   (b) Else, set $i = i + 1$, update the guess: $c_0 = c$.

Secant Method

The disadvantage of the Newton method is that sometimes, evaluating the derivative can be challenging or analytically impossible. In such a case, we can approximate it by using linear approximation

$$f'(x_{n-1}) = \lim_{x \to x_{n-1}} \frac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}}.$$

Given 2 initial guesses x_0, x_1, we can iterate to find the root. This method is very similar to Newton's method above. The main difference is that in step 2, we will use approximated values. Also, the secant method requires 2 initial guesses instead of 1.

1. Provide two initial guesses c_0, c_1.
2. Find $f(c_0)$ and $f(c_1)$.
3. Update the guess (look at Eq.(13)).

$$c = c_1 - f(c_1)\frac{c_1 - c_0}{f(c_1) - f(c_0)}.$$

4. Calculate $\varepsilon = |c - c_0|$:
   (a) if $\varepsilon < tol$, end the program and output c.
   (b) Else, set $i = i + 1$, $c_1 = c$.

Exercise 2. For each algorithm above, do the following:

1. Write a simple loop, following the algorithms to find the root. Test with the following function

$$f(x) = \cos(x) - x^3 + 1.$$

   In Julia, just type `cos(x) - x^ 3`. Correct answer: 1.12656.

2. Make it into a function.

Built-in Program

The easiest way is to use built-in functions. Of course, you need to know the syntax.

```
// Python code example
from scipy.optimize import fsolve

def func_k(k, beta, alpha, A, n):
        F = k - beta * A * (1-alpha) * (k**alpha) / ((1+n)*(1+beta))
        return F

k_guess = 2
sol_k = fsolve(func_k, k_guess, args=(beta, alpha, A, n))

kstar = sol_k[0]
```

```
// Julia code example
using NLsolve

function func_k!(F, k, beta, A, alpha, n)
    F[1] = k[1] - beta*A*(1-alpha)*(k[1]^alpha) / ((1+n)*(1+beta))
end

k_guess = [2]
sol = nlsolve((F, k) -> func_k!(F, k, beta, A, alpha, n), k_guess)

kstar = sol.zero[1]
```

Gauss-Seidel Algorithm (Fixed Point Iteration)

If your dynamics have a fixed point k^* and it is stable, i.e.

$$|\phi'(k^*)| < 1 \text{ given } k_{t+1} = \phi(k_t).$$

then, by successive iteration of a given initial value, you will get to that fixed point.

Algorithm 1 (Gauss-Seidel)**.** To solve a simple OLG with one state variable.

1. Guess the initial value of k_0
2. Calculate other endogenous variables w, R based on (6) and (7)
3. Solve optimal saving decision s based on (2).
4. Calculate again the capital-labor ratio and get k_1 based on (8).
5. Calculate the error and verify if the algorithm has converged

$$\text{error} = \frac{k_1 - k_0}{k_0}.$$

If error > 0, update the capital-labor ratio with $0 < \lambda < 1$ as the update parameter

$$k_{0,new} = \lambda k_1 + (1 - \lambda) k_0.$$

and repeat step 2. Otherwise, if error $= 0$, stop.

```
# some functions to calculate at step 2
function func_w(k)
    return (1 - alpha) * A * (k^alpha)
end
function func_s(k)
    w = func_w(k)
    return beta * w / (1 + beta)
end
function func_k(k)
    s = func_s(k)
    return func_s(k) / (1 + n)
end
# loop preparation
lamda = 0.5                # update parameter
```

```
tol = 1e-6                # threshold (close to 0)
max_iter = 100
error = 1.0
# initial guess
k = 2.0
# loop
iter = 1
while error > tol && iter < max_iter
    k_new = lamda * func_k(k) + (1 - lamda) * k
    error = abs(k_new - k)
    k = k_new
    iter += 1
end

println("kstar: ", k)
println("Number of iterations: ", iter)
```

After 33 iterations, the algorithm also gives us the same result for the steady state at 1e-6 tolerance error.

1.3 Backward-looking Transition Dynamics

In this section, we follow the definition of an intertemporal equilibrium. Sometimes, when we simulate a change in parameters or policy, together with calculating the new steady state, investigating the transition dynamics can also be interesting, especially in welfare comparison.

In this example, we simulate the situation when population growth reduces from 0.3 to 0.2. Assuming that the economy is initially at the previous steady state, we want to see how capital evolves given a sudden change in n.

```
n = 0.2
T = 8                          # number periods to simulate
knew = zeros(T)                  # an array of new values of k
time = zeros(T)                  # an array of time
knew[1] = k                  # initial k (calculated previously)
for t in 2:T
    knew[t] = func_k(knew[t-1])
    time[t] = t
end
using Plots
plot(time, knew, xlabel="Time", ylabel="k", title="Transition of
↪  k(t)")
```

Exercise 3. In this exercise, you need to solve for the steady state of Eq.(10).

1. Write 4 programs to show how to solve for the steady state in 4 different ways. Please use Julia.

2. Provide the same initial guess (for the second, set the second guess to 1.5 times the first), and calculate how many iterations each method requires.

3. At the steady state, assume that ρ increases to -2. Derive the new steady state and transition dynamics for 10 periods.