INSEIKAI Tohoku BootCamp 2024
Mathematics IV
Computational Macroeconomics

Quang-Thanh Tran *

Graduate School of Economics and Management
Tohoku University

June 27, 2024

Contents

*tran.quang.thanh.p1@dc.tohoku.ac.jp

Preface

Most of the current macroeconomic models cannot be solved by hand when the agents in the model live longer than 2 periods. In those cases, numerical methods are necessary. In this camp, we will learn how to solve these models in their most basic forms and derive the solutions via computational methods.

Requirements

1. Being comfortable with basic calculus (mostly differentiation).
2. Know at least some programming, such as Julia and Python. (basic level is sufficient). You can use Excel or MATLAB, but they are not free.

Schedule

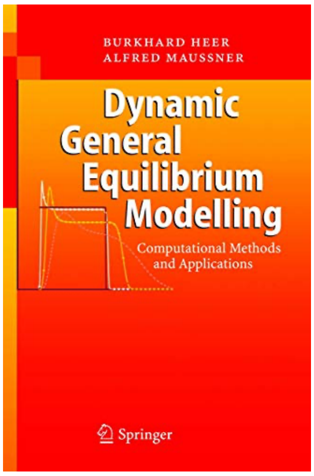
In this course, we closely follow chapters 1, 2, and 9 from the celebrated Heer and Maußner (2009). We will use Julia and Python as the main programming languages.

1. First, we study the simple OLG model to practice coding and solution concepts.
2. We study the infinite-horizon Ramsey model. Here, you will learn that solutions are not easy to obtain due to the infinite lifetime. To pin down the policy function, we use value function iteration and perturbation methods.
3. We study a finite-horizon OLG model. Here, we can use backward induction to solve the model because lifetime is finite. To pin down the policy function, we use the direct computation method.

Materials

We will occasionally visit materials from their other editions (Heer and Maußner, 2005, 2024) and other books about DSGE such as Costa (2018); Torres (2020) and large-scale OLG modeling such as Heer (2019); Feldstein and Auerbach (2002a,b). Other useful online materials include Eric Sims' Ph.D. Macro Theory II and Vermandel's DSGE Dynare Model Matlab Codes.

You can also find many valuable materials here: `https://juejung.github.io/jdocs/Comp/html/Slides_OLG_I.html`. If you have problems with coding, refer to the following cheat sheet (`https://cheatsheets.quantecon.org/index.html`)



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 $R_t(k_t)$.

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

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.

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 $c = c_0 - \dfrac{f(c_0)}{f'(c_0)}$
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.

2 Infinite-Horizon Representative Agent Model

Premise

1. There is one representative agent who lives forever. This is justified when you think of agents as households. Members are periodically born and die, but households sustain and, therefore, become infinite.
2. There is no retirement, as agents live forever. They provide capital and labor and consume goods every period.
3. Population is normalized to 1.

2.1 Ramsey Model

Household problem

$$\max_{c_t, k_{t+1}} \sum_{t=0}^{\infty} \beta^t u(c_t)$$

subject to

$$\begin{aligned} c_t + k_{t+1} &= f(k_t) + (1-\delta)k_t, \\ k_0 &> 0. \end{aligned}$$

Exercise 4. Solve for the FOC (in this case, the FOC is the Euler)

1. With Lagrangian

$$\mathcal{L} = \sum_{t=0}^{\infty} \beta^t u(c_t) + \sum_{t=0}^{\infty} \lambda_t \left[f(k_t) + (1-\delta)k_t - k_{t+1} - c_t\right]$$

Grouping all the summation and rewriting the Lagrangian:

$$\mathcal{L} = \sum_{t=0}^{\infty} \left[\beta^t u(c_t) + \lambda_t \left(f(k_t) + (1-\delta)k_t - k_{t+1} - c_t\right)\right]$$

The term inside the sum is optimized at each t following the modified Lagrangian:

$$\mathcal{L} = \beta^t u(c_t) + \lambda_t \left(f(k_t) + (1-\delta)k_t - k_{t+1} - c_t\right)$$

FOC: (Note that k_{t+1} appears twice at time t and $t+1$)

$$(c_t) : \frac{\partial \mathcal{L}}{\partial c_t} = \beta^t u'(c_t) - \lambda_t = 0 \tag{14}$$

$$(k_{t+1}) : \frac{\partial \mathcal{L}}{\partial k_{t+1}} = -\lambda_t + \lambda_{t+1}\left[f'(k_{t+1}) + (1-\delta)\right] = 0 \tag{15}$$

By virtue of Eq. (14), we see that:

$$\beta^{t+1} u'(c_{t+1}) = \lambda_{t+1}$$

Plugging back to Eq. (15) and rearranging give us the Euler equation:

$$\frac{u'(c_t)}{u'(c_{t+1})} = \beta\left[f'(k_{t+1}) + (1-\delta)\right]$$

For sufficient, the following transversality condition holds [2]

$$\lim_{T\to\infty} \beta^T u'(c_T) k_{T+1} = 0.$$

The intuition of the transversality condition is partly that "there is no savings in the last period". But as there is no "last period" in an infinite horizon environment, we take the limit as time goes to infinity. The slackness condition ensures that

2. With Bellman equation

Assume that we have found the optimized sequence of capital holding $\{k_t\}_{t=0}^{\infty}$, then the value of lifetime utility associated with that optimum is

$$V(k_t) = \max_{\{c_t, k_{t+1}\}_{t=0}^{+\infty}} \sum_{t=0}^{+\infty} \beta^t u(c_t)$$

where

$$c_t = f(k_t) - k_{t+1} + (1-\delta)k_t$$

We can write it in recursive form

$$V(k_t) = \max_{k_{t+1}} \left[u(f(k_t) - k_{t+1} + (1-\delta)k_t) + \beta V(k_{t+1})\right]$$

Choosing c_t is the same as choosing k_{t+1}. Maximizing the Value function wrt. the control variable k_{t+1}:

$$\begin{aligned}
&\frac{\partial V(k_t)}{\partial k_{t+1}} = 0 \\
&\Leftrightarrow \frac{\partial u(k_{t+1})}{\partial k_{t+1}} + \beta \frac{\partial V(k_{t+1})}{\partial k_{t+1}} = 0 \\
&\Leftrightarrow -u'(c_t) + \beta \frac{\partial V(k_{t+1})}{\partial k_{t+1}} = 0
\end{aligned}$$

By the Envelope theorem, we obtain the Benveniste-Scheinkman Equation

$$\frac{\partial V(k_t)}{\partial k_t} = (f'(k_t) + 1 - \delta) u'(f(k_t) - k_{t+1} + (1-\delta)k_t),$$

Forwarding 1 period

$$\begin{aligned}
\frac{\partial V(k_{t+1})}{\partial k_{t+1}} &= (f'(k_{t+1}) + 1 - \delta) u'(f(k_{t+1}) - k_{t+2} + (1-\delta)k_{t+1}) \\
&= (f'(k_{t+1}) + 1 - \delta) u'(c_{t+1})
\end{aligned}$$

Derive the Euler equation relating the dynamics of the choice variable.

$$\frac{u'(c_t)}{u'(c_{t+1})} = \beta(f'(k_{t+1}) + (1-\delta))$$

Assume the following functional form

$$\begin{aligned}
u(c_t) &= \ln c_t, \\
f(k_t) &= k_t^{\alpha}.
\end{aligned}$$

[2] to see why, you can visit: https://economics.stackexchange.com/questions/15290/transversality-condition-in-neoclassical-growth-model

Exercise 5 (Solve for the steady state)**.** At the steady states

$$c_t = c_{t+1} = \bar{c},$$
$$k_t = k_{t+1} = \bar{k}$$

Using the Euler, prove that

$$\bar{k} = \left(\frac{\alpha\beta}{1-(1-\delta)\beta}\right)^{1/(1-\alpha)}, \tag{16}$$
$$\bar{c} = \left(\frac{1-\beta[1-(1-\alpha)\delta]}{\alpha\beta}\right)\left(\frac{\alpha\beta}{1-(1-\delta)\beta}\right)^{1/(1-\alpha)}.$$

Thus far, we have only solved the Euler equation and the steady states. However, the solution is a sequence of $\{c_t, k_t\}_{t=0}^{\infty}$ that solves the lifetime utility. To derive the sequence, we need to pin down the policy function

$$k_{t+1} = h(k_t).$$

The mission is to estimate this $h(\cdot)$ function. Below, we introduce some solution methods to find such a function.

Parameters

Parameters	Value
β	0.99
α	0.3
δ	0.1

2.2 Method of Undetermined Coefficients

First, we need to guess the functional form of the Value function using some parameters. Assume that

$$V = a + b\ln(k)$$

with a and b is yet undetermined coefficients. We can rewrite the recursive form as

$$\max_{k'} \ln(f(k) - k' + (1-\delta)k) + \beta(a + b\ln k')$$

The FOC wrt k' is

$$k' = \frac{\beta b}{1+\beta b}k^{\alpha}.$$

Plugging back to the value function to derive a and b. This method is very limited as the guess must be correct, and the function is analytically differentiable. We derive below some alternatives. The first is a local solution called the perturbation method, and the second is a global solution method known as value function and policy function iteration.

Exercise 6. Write a program that solves and plots this policy function.

2.3 Pertubation Methods: Linear Approximation

In previous methods, the exact solution to the policy function could be obtained. However, this is not always feasible. In some cases, approximation is preferred as it provides faster computation, especially in models with stochastic elements.

Taylor Approximation

Consider the following case system

$$\begin{aligned} x_{t+1} &= f(x_t, y_t) \\ y_{t+1} &= g(x_t, y_t) \end{aligned} \tag{17}$$

Consider the linear dynamics in $\mathbb{R}^2 \to \mathbb{R}^2$. Given the initial state (x_0, y_0). Assume that

$$\begin{aligned} \bar{x} &= f(\bar{x}, \bar{y}), \\ \bar{y} &= g(\bar{x}, \bar{y}) \end{aligned} \tag{18}$$

be the steady state $(\bar{x}, \bar{y})$ of the system (17). The first-order Taylor expansion of $f(\cdot)$ around a steady state:

$$f(x, y) - f(\bar{x}, \bar{y}) \approx f'_x(\bar{x}, \bar{y})(x - \bar{x}) + f'_y(\bar{x}, \bar{y})(y - \bar{y})$$

Similarly for $g(\cdot)$:

$$g(x, y) - g(\bar{x}, \bar{y}) \approx g'_x(\bar{x}, \bar{y})(x - \bar{x}) + g'_y(\bar{x}, \bar{y})(y - \bar{y})$$

From (17),(18), we can write them in matrix form [3] as

$$\begin{pmatrix} x_{t+1} - \bar{x} \\ y_{t+1} - \bar{y} \end{pmatrix} = \underbrace{\begin{pmatrix} f'_x(\bar{x}, \bar{y}) & f'_y(\bar{x}, \bar{y}) \\ g'_x(\bar{x}, \bar{y}) & g'_y(\bar{x}, \bar{y}) \end{pmatrix}}_{\mathbf{J}} \begin{pmatrix} x_t - \bar{x} \\ y_t - \bar{y} \end{pmatrix} \tag{19}$$

where, as we all know by now, $\mathbf{J}$ is the Jacobian matrix. The system has been "linearized" and can be analyzed similarly to the linear case.

Linear Approximation of Saddle Path

From the resource constraint and Euler equation

$$\begin{aligned} k_{t+1} + c_t &= f(k_t) + (1 - \delta)k_t, \\ u'(c_t) &= \beta u'(c_{t+1})[f'(k_{t+1}) + (1 - \delta)] \end{aligned}$$

At the steady state

$$\begin{aligned} \bar{c} &= f(\bar{k}) - \delta\bar{k}, \\ 1/\beta &= f'(\bar{k}) + (1 - \delta) \end{aligned}$$

We can write the behavior of variables near the steady state as

$$\begin{pmatrix} k_{t+1} - \bar{k} \\ c_{t+1} - \bar{c} \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} k_t - \bar{k} \\ c_t - \bar{c} \end{pmatrix} \tag{20}$$

[3] This is called to "linearize" around the steady state

We want to estimate A, B, C, D. Near the steady state $(\bar{c}, \bar{k})$ we have

$$\begin{aligned} c_t &= \bar{c} + (c_t - \bar{c}), \\ f(k_t) + (1-\delta)k_t &\approx f(\bar{k}) + (1-\delta)\bar{k} + (f'(\bar{k}) + (1-\delta))(k_t - \bar{k}) \\ k_{t+1} &\approx \bar{k} + (k_t - \bar{k}). \end{aligned}$$

Substituting these into the resource constraint

$$k_{t+1} + \bar{c} + (c_t - \bar{c}) = f(\bar{k}) + (1-\delta)\bar{k} + (f'(\bar{k}) + (1-\delta))(k_t - \bar{k})$$

Rearranging

$$(k_{t+1} - \bar{k}) = (1/\beta)(k_t - \bar{k}) - (c_t - \bar{c}) \tag{21}$$

Next, we need one more equation containing $c_{t+1} - \bar{c}$, let us use the **Euler equation** and log-linearize it

$$\ln u'(c_t) - \ln u'(c_{t+1}) - \ln \beta = \ln[f'(k_{t+1}) + (1-\delta)] \tag{22}$$

At the steady state, since $c_{t+1} = c_t = \bar{c}$

$$\ln u'(\bar{c}) - \ln u'(\bar{c}) - \ln \beta = \ln[f'(\bar{k}) + (1-\delta)] \tag{23}$$

Subtract Eq.(23) from (22) to obtain

$$(\ln u'(c_t) - \ln u'(\bar{c})) - (\ln u'(c_{t+1}) - \ln u'(\bar{c})) = \ln[f'(k_{t+1}) + (1-\delta)] - \ln[f'(\bar{k}) + (1-\delta)] \tag{24}$$

Near the steady state $(\bar{c}, \bar{k})$ we have

$$\begin{aligned} \ln u'(c_{t+1}) - \ln u'(\bar{c}) &\approx \frac{u''(\bar{c})}{u'(\bar{c})}(c_{t+1} - \bar{c}), \\ \ln u'(c_t) - \ln u'(\bar{c}) &\approx \frac{u''(\bar{c})}{u'(\bar{c})}(c_t - \bar{c}), \\ \ln[f'(k_{t+1}) + (1-\delta)] - \ln[f'(\bar{k}) + (1-\delta)] &\approx [\ln(f'(\bar{k}) + (1-\delta))]'(k_{t+1} - \bar{k}) \\ &= \frac{f''(\bar{k})}{f'(\bar{k}) + (1-\delta)}(k_{t+1} - \bar{k}) \\ &= \beta f''(\bar{k})(k_{t+1} - \bar{k}) \end{aligned}$$

Plugging back to (24) yields

$$\frac{u''(\bar{c})}{u'(\bar{c})}(c_t - \bar{c}) - \frac{u''(\bar{c})}{u'(\bar{c})}(c_{t+1} - \bar{c}) = \beta f''(\bar{k})(k_{t+1} - \bar{k}) \tag{25}$$

Rearranging

$$-\beta f''(\bar{k})(k_{t+1} - \bar{k}) - \frac{u''(\bar{c})}{u'(\bar{c})}(c_{t+1} - \bar{c}) = -\frac{u''(\bar{c})}{u'(\bar{c})}(c_t - \bar{c}) \tag{26}$$

Combining (21) and (26) yields the system

$$\begin{aligned} (k_{t+1} - \bar{k}) + 0 \cdot (c_{t+1} - \bar{c}) &= (1/\beta)(k_t - \bar{k}) - (c_t - \bar{c}), \\ -\beta f''(\bar{k})(k_{t+1} - \bar{k}) - \frac{u''(\bar{c})}{u'(\bar{c})}(c_{t+1} - \bar{c}) &= 0 \cdot (k_t - \bar{k}) - \frac{u''(\bar{c})}{u'(\bar{c})}(c_t - \bar{c}) \end{aligned}$$

Write this in matrix form

$$\begin{pmatrix} 1 & 0 \\ -\beta f''(\bar{k}) & -\frac{u''(\bar{c})}{u'(\bar{c})} \end{pmatrix} \begin{pmatrix} k_{t+1} - \bar{k} \\ c_{t+1} - \bar{c} \end{pmatrix} = \begin{pmatrix} 1/\beta & -1 \\ 0 & -\frac{u''(\bar{c})}{u'(\bar{c})} \end{pmatrix} \begin{pmatrix} k_t - \bar{k} \\ c_t - \bar{c} \end{pmatrix}$$

To transform it into a form similar to (20), we premultiply both sides with the inverse of the first matrix and obtain

$$\begin{aligned} \begin{pmatrix} k_{t+1} - \bar{k} \\ c_{t+1} - \bar{c} \end{pmatrix} &= \begin{pmatrix} 1 & 0 \\ -\beta f''(\bar{k}) & -\frac{u''(\bar{c})}{u'(\bar{c})} \end{pmatrix}^{-1} \begin{pmatrix} 1/\beta & -1 \\ 0 & -\frac{u''(\bar{c})}{u'(\bar{c})} \end{pmatrix} \begin{pmatrix} k_t - \bar{k} \\ c_t - \bar{c} \end{pmatrix} \\ &= \underbrace{\begin{pmatrix} 1/\beta & -1 \\ \frac{u'(\bar{c}) f''(\bar{k})}{u''(\bar{c})} & 1 + \beta \frac{u'(\bar{c}) f''(\bar{k})}{u''(\bar{c})} \end{pmatrix}}_{\mathbf{J}} \begin{pmatrix} k_t - \bar{k} \\ c_t - \bar{c} \end{pmatrix} \end{aligned}$$

This Jacobian $\mathbf{J}$ has two eigenvalues λ_1, λ_2 satisfying

$$\begin{aligned} \det \mathbf{J} &= 1/\beta = \lambda_1 \lambda_2, \\ tr\mathbf{J} &= 1 + \frac{1}{\beta} + \frac{\beta u'(\bar{c}) f''(\bar{k})}{u''(\bar{c})} = \lambda_1 + \lambda_2 = \triangle \end{aligned}$$

Since $u'' < 0, f'' < 0$, we can say that

$$|1 + \det \mathbf{J}| = 1 + 1/\beta < tr\mathbf{J} = 1 + 1/\beta + \frac{\beta u'(\bar{c}) f''(\bar{k})}{u''(\bar{c})}$$

The steady state is a saddle point, implying that $\lambda_1 < 1 < \lambda_2$. The smaller root is stable, while the bigger root is unstable. We can extract $\lambda_1 = \triangle - \lambda_2$. Then

$$\begin{aligned} \det \mathbf{J} &= \lambda_2(\triangle - \lambda_2) \\ \iff \triangle &= \frac{\det \mathbf{J}}{\lambda_2} + \lambda_2 \end{aligned}$$

Hence, λ_1, λ_2 are the solutions of the following quadratic function

$$\phi(\lambda) = \lambda^2 - \triangle \lambda + 1/\beta$$

The eigenvector associated with the eigenvalues are

$$(\lambda_1) : \begin{pmatrix} v_{11} \\ v_{12} \end{pmatrix}, \qquad (\lambda_2) : \begin{pmatrix} v_{21} \\ v_{22} \end{pmatrix}$$

Define a new matrix

$$V = \begin{pmatrix} v_{11} & v_{21} \\ v_{12} & v_{22} \end{pmatrix}$$

with its inverse matrix

$$V^{-1} = \frac{1}{v_{11} v_{22} - v_{12} v_{21}} \begin{pmatrix} v_{22} & -v_{21} \\ -v_{12} & v_{11} \end{pmatrix}$$

The eigen diagonal matrix is

$$\Lambda = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

Then, we can eigendecompose the matrix $\mathbf{J}$ to

$$\mathbf{J} = V\Lambda V^{-1}$$

The system is now written as

$$\begin{pmatrix} k_{t+1} - \bar{k} \\ c_{t+1} - \bar{c} \end{pmatrix} = V\Lambda V^{-1} \begin{pmatrix} k_t - \bar{k} \\ c_t - \bar{c} \end{pmatrix}$$

Iterating forward, starting from initial c_0, k_0 implies

$$\begin{pmatrix} k_t - \bar{k} \\ c_t - \bar{c} \end{pmatrix} = V\Lambda^t V^{-1} \begin{pmatrix} k_0 - \bar{k} \\ c_0 - \bar{c} \end{pmatrix}$$

Writing out explicitly

$$\begin{aligned} k_t - \bar{k} &= v_{21} \frac{v_{22}(c_0 - \bar{c}) - v_{12}(k_0 - \bar{k})}{v_{11}v_{22} - v_{12}v_{21}} \lambda_1^t - v_{22} \frac{v_{21}(c_0 - \bar{c}) - v_{11}(k_0 - \bar{k})}{v_{11}v_{22} - v_{12}v_{21}} \lambda_2^t \\ c_t - \bar{c} &= v_{11} \frac{v_{22}(c_0 - \bar{c}) - v_{12}(k_0 - \bar{k})}{v_{11}v_{22} - v_{12}v_{21}} \lambda_1^t - v_{12} \frac{v_{21}(c_0 - \bar{c}) - v_{11}(k_0 - \bar{k})}{v_{11}v_{22} - v_{12}v_{21}} \lambda_2^t \end{aligned}$$

If λ_2 is the unstable root, set

$$c_0 - \bar{c} = \frac{v_{11}}{v_{21}}(k_0 - \bar{k})$$

will neutralize the unstable root (explosive dynamics). That's why consumption is also called the jump variable. Plugging it back to the initial consumption yields

$$c_t - \bar{c} = v_{11} \frac{v_{22}\frac{v_{11}}{v_{21}} - v_{12}}{v_{11}v_{22} - v_{12}v_{21}} \lambda_1^t (k_0 - \bar{k}) = \frac{v_{11}}{v_{21}} \lambda_1^t (k_0 - \bar{k})$$

and so the capital accumulation

$$k_t - \bar{k} = v_{21} \frac{v_{22}\frac{v_{11}}{v_{21}} - v_{12}}{v_{11}v_{22} - v_{12}v_{21}} \lambda_1^t (k_0 - \bar{k}) = \lambda_1^t (k_0 - \bar{k})$$

Hence, the solution of k_t is

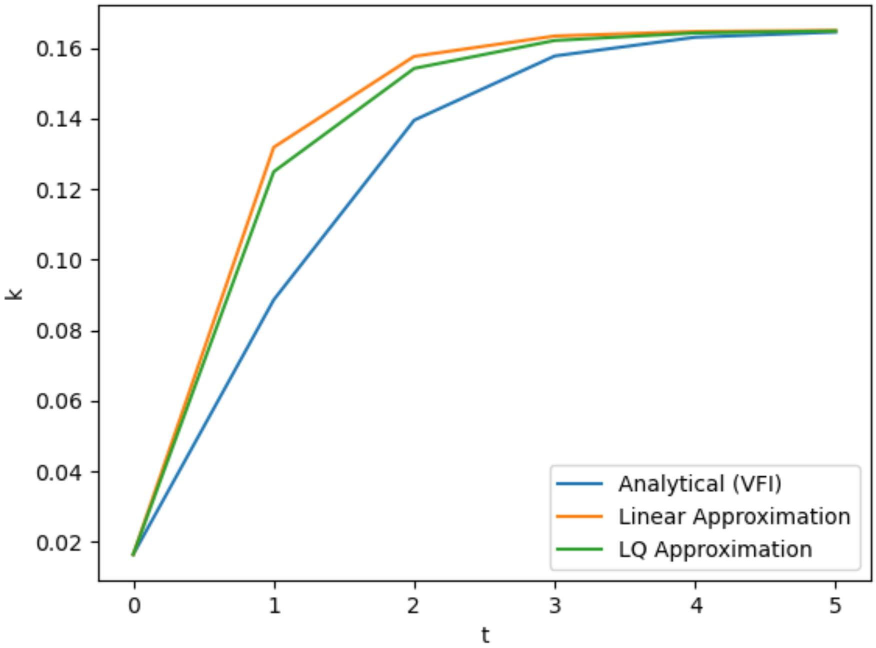
$$k_{t+1} - \bar{k} = \lambda_1 (k_t - \bar{k})$$

where λ_1 is the stable root.

Exercise 7. Compare:

1. Write a program that solves the policy function using this linear approximation.

There are also other perturbation methods, one commonly used is the LQ method. It was proposed by Hansen and Prescott (1995). L means linearity of constraint, and Q means quadratic utility function. We must look for a linear law of motion and put all remaining nonlinear relations into the current consumption. We do not study that method here, but here is a comparison for accuracy.



2.4 Value Function Iteration

The above methods are called local methods as they try to approximate the policy function point-by-point. In this section, we use one global method called Value Function Iteration. First, we do it by hand. You will then see how the algorithm works. Doing it by hand also delivers an analytical solution so you can check with the method of undetermined coefficients. Later, we write a code that does the iteration for us.

First, drop t notation for short and use a "prime" to denote variables at $t+1$. The true value function is the limit of the following

$$V^{s+1}(k) = \max_{k'} u(f(k) - k' + (1-\delta)k) + \beta V^s(k')$$

1. Initial guess: $V^0 = 0$.
2. $s = 1$: with V^0 known, we find k' that maximizes $V^1(k)$, then substitute this k' back to V^1 to derive the value of V^1.
3. $s = 2$: with V^1 known previously, we find k' that maximizes $V^2(k)$, then substitute this k' back to V^2 to derive the value of V^2.
4. iterate as many as you can until you see the pattern.
5. take the limits of $s \to \infty$.

The method works because Ramsey's value function is a contraction mapping (Acemoglu, 2008, p.190-194). You should be able to derive

$$k' = \alpha\beta k^\alpha$$

Now, we use a computer to perform this iteration based on the following procedure.

1. Choose a grid that must contain the steady state. The grid should contain a steady state. The steady-state satisfies

$$f'(\bar{k}) = \frac{1}{\beta} \Rightarrow \bar{k}$$

We also want to find the maximal sustainable capital stock (consume nothing)

$$f(\hat{k}) = \hat{k} \Rightarrow \hat{k}$$

The minimum grid point should be larger than 0, and the maximum grid point minimum than $\hat{k}$. We generate n points equally spaced on this grid, indexed by i.

2. Initiate an array of initial guesses

$$\text{(naive) } V^0 = 0,$$
$$\text{(smart) } V^0 = \frac{u(\bar{c})}{1-\beta}$$

3. For each point $k(i)$, find a $k(j)$ on the grid that maximizes[4]

$$V_i^{s+1} = \sup_{k_j \in kgrid} u(f(k_i) - k_j) + \beta V_j^s$$

substitute k_j back to V_i^{s+1} to derive the value V^1.

Update V^0 to V^1, and store the value of optimal k_j.

4. For each iteration s, check the error $|V^0 - V^1|$. If it is smaller than tol, stop. Otherwise, go back to step 3.

Exercise 8. Write a program to solve the model based on value function iteration.

A good reference source: `https://www.eco.uc3m.es/~jrincon/Teaching/Master/SDDP.pdf#page=8.70`. Check page 21.

The sample code is found in the Appendix. In the sample code, we use naive guesses. In your implementation, try to use smart guesses.

The value function iteration is a slow process, as the convergence rate is β. There are other iterative methods that produce faster convergence, such as policy function iteration or Euler equation iteration. You can see some examples of such algorithms in the Appendix.

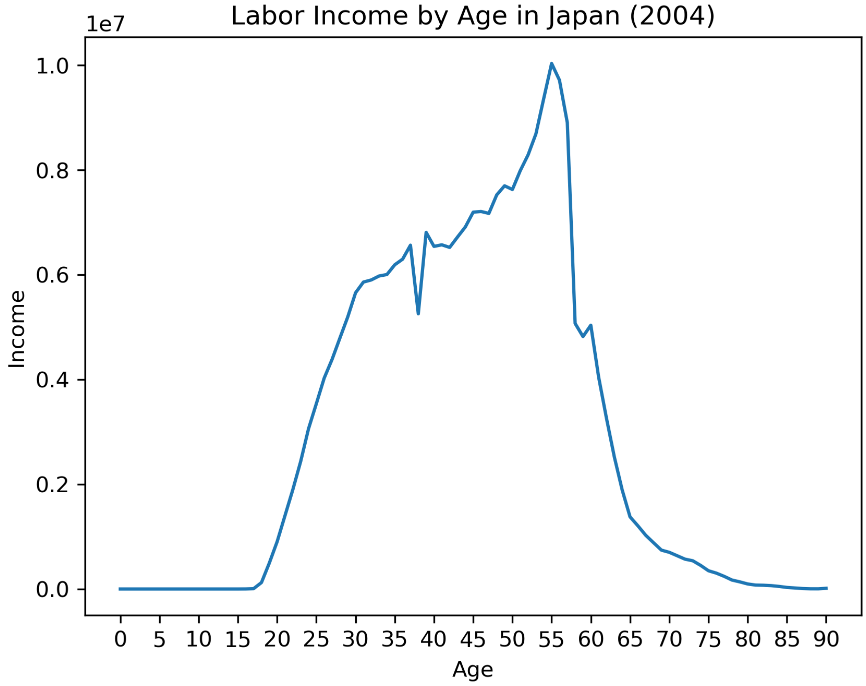
[4]or use the Binary Search algorithm. Basically, it searches on the grid and returns the index of the point that maximizes the objective function.

3 Large-scale OLG

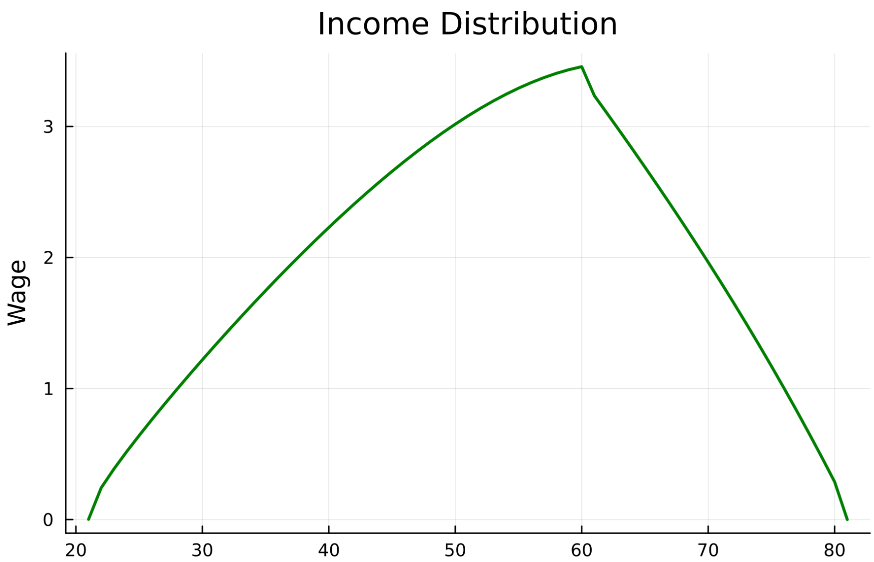
Premise

1. Agents are heterogeneous in age.
2. There is retirement. Agents do not live forever.
3. Population is constant forever.

We want to reproduce a life cycle profile such as this



Our goal is to capture similar qualitative features in the model.



3.1 The Model

Building Blocks

Age structure: The representative household's life span is 60, such that

$$T + TR = 40 + 20 = 60$$

where T is the working length and TR is the retirement length. Labor supply n_t^s follows

$$l_t^s = 1 - n_t^s \text{ for } t \in \{1, 2, \dots, 40\} \tag{27}$$
$$l_t^s = 1 \text{ for } t \in \{41, 42, \dots, 60\} \tag{28}$$

where l_t^s is leisure. After T years, retirement is mandatory. The agent's maximization problem is

$$\sum_{s=1}^{T+TR} \beta^{s-1} u(c_{s+t-1}^s, l_{t+s-1}^s) \tag{29}$$

where β is the discount factor. The instantaneous utility function:

$$u(c, l) = \frac{((c + \psi)l^{\gamma})^{1-\eta} - 1}{1 - \eta} \tag{30}$$

An agent is born without wealth and leaves no bequests upon death, thus $k_t^1 = k_t^{61} = 0$. The real budget constraint of the working agent is given by

$$k_{t+1}^{s+1} = (1 + r_t)k_t^s + (1 - \tau_t)w_t n_t^s - c_t^s \text{ for } s = 1, \dots, T \tag{31}$$

where r_t, w_t are the interest and wage rates, while τ is the social security contribution tax. Once retired, the agents receive public pensions b and no labor earnings. The budget constraint for a retiree is

$$k_{t+1}^{s+1} = (1 + r_t)k_t^s + b - c_t^s \text{ for } s = T + 1, \dots, TR \tag{32}$$

Production is Cobb-Douglas technology:

$$Y_t = N_t^{1-\alpha} K_t^{\alpha}.$$

Let $\delta \in [0, 1]$ be the depreciation rate. The factor prices are

$$w_t = (1 - \alpha)K_t^{\alpha} N_t^{1-\alpha},$$
$$r_t = \alpha K_t^{\alpha-1} N_t^{-\alpha} - \delta.$$

Its budget is balanced every period such that

$$\tau w_t N_t = \frac{TR}{T + TR} b.$$

Equilibrium

To derive the equilibrium, we need to solve the household problem.

Exercise 9. Solve a young household's problem by maximizing (29), using the functional form at (30), with respect to (27) and (31) by doing the following steps

1. Write the Lagrangian.
2. Derive the FOC for c and l.
3. Derive the Euler equation.

Solving a retiree household's problem by maximizing (29), using the functional form at (30), with respect to (28) and (32).

1. From (28), we know that $n_t^s = 0$.
2. Derive the Euler equation

Answer 1. The FOC:

$$\frac{u_l(c_t^s, l_t^s)}{u_c(c_t^s, l_t^s)} = \gamma \frac{c_t^s + \psi}{l_t^s} = (1 - \tau_t) w_t \tag{33}$$

The Euler:

$$\frac{1}{\beta} = \frac{u_c(c_{t+1}^{s+1}, l_{t+1}^{s+1})}{u_c(c_t^s, l_t^s)}(1 + r_{t+1}) = \frac{(c_{t+1}^{s+1} + \psi)^{-\eta}(l_{t+1}^{s+1})^{\gamma(1-\eta)}}{(c_t^s + \psi)^{-\eta}(l_t^s)^{\gamma(1-\eta)}}(1 + r_{t+1}). \tag{34}$$

We can also represent the households' problem recursively. Let $V^s(k_t^s, K_t, N_t)$ be the value of the objective function of s-year old agent with wealth k_t^s, K_t, N_t. It is defined as the solution to the dynamic program

$$V^s(k_t^s, K_t, N_t) = \begin{cases} \max\limits_{k_{t+1}^{s+1}, c_t^s, l_t^s} u(c_t^s, l_t^s) + \beta V^{s+1}(k_{t+1}^{s+1}, K_{t+1}, N_{t+1}) \text{ for } s = 1, \dots, T \\ \max\limits_{k_{t+1}^{s+1}, c_t^s} u(c_t^s, 1) + \beta V^{s+1}(k_{t+1}^{s+1}, K_{t+1}, N_{t+1}) \text{ for } s = T+1, \dots, T+TR-1 \end{cases} \tag{35}$$

subject to Eq.(31) and (32), respectively. Furthermore

$$V^{T+TR}(k_t^{T+TR}, K_t^{T+TR}, N_t^{T+TR}) = u(c^{T+TR}, 1)$$

implying that agents obtain nothing if they save after they die. The recursive formulation is useful for the Value Function Iteration method.

Definition 2. Equilibrium

1. Individual and aggregate behaviors are consistent

$$N_t = \sum_{s=1}^{T+TR} \frac{n_t^s}{T+TR}, \tag{36}$$

$$K_t = \sum_{s=1}^{T+TR} \frac{k_t^s}{T+TR} \tag{37}$$

The formulation means that aggregate labor (capital) supply is equal to the sum of supplies of each cohort, weighted by its mass $1/(T+TR) = 1/60$.

2. Goods market clear

$$N_t^{1-\alpha} K_t^{\alpha} = \sum_{s=1}^{T+TR} \frac{c_t^s}{T+TR} + K_{t+1} - (1-\delta) K_t$$

3. Given $\{w_t, r_t, b, \tau\}$, the household's problem is solved according to Eq.(33) and (34), the firm's optimization is solved, and the government budget is balanced.

3.2 Steady State

The concept of a steady state can be characterized by a constant distribution of capital stock over generations.

$$\{k_t^s\}_{s=1}^{60} = \{k_{t+1}^s\}_{s=1}^{60} = \{\bar{k}^s\}_{s=1}^{60}$$

As a consequence, every other variable, such as r, w, b, τ, also becomes a constant.

The computation of the steady states is more complex than Section 1. However, since our functions are well-behaved, we can use the simple iteration method to reach the steady state by following the algorithm below.

1. Make initial guesses of the steady state values of K and N.
2. Compute w, r, τ, b that solve the firm's problem and the government's budget set.
3. Compute the optimal path for consumption, savings, and labor supply by backward iteration.
    (a) We know $k^1 = k^{61} = 0$. Make a guess of k^{60}.
    (b) With k^{61}, k^{60} known, solve for $k^{59}, k^{58}, \ldots, k^2, k^1$.
    (c) Compute k^1:
        i. if $k^1 = 0$. Stop the loop and output the series of $k^1, \ldots, k^{60}$.
        ii. else, if $k^1 \neq 0$, update k^{60} using the secant method.
4. Recompute the new aggregate K and N.
5. If they are the same as the initial guess. Stop. Otherwise, update a new K and N and go back to step 2 until convergence.

Parameters

Parameters	Value
β	0.98
η	2
α	0.36
δ	0.1
γ	2
ψ	0.001

To calculate the tax rate, set the replacement ratio $\xi = 0.3$ such that

$$\xi = \frac{b}{(1-\tau)w\bar{n}}$$

with $\bar{n}$ is the average labor supply, which equals to $N(T + TR)/T$. Since we want a realistic value, you can set the target values for the steady states of

$$\bar{n} = 0.35, \qquad \tau = \frac{\xi}{2+\xi}, \qquad r = 0.045$$

Given N and r, we can calculate w and K, then solve for b, which completes step 2.

3.3 Direct Computation

Exercise 10. In this exercise, you are asked to perform direct computation at each age. Note that there are three crucial age thresholds: age 39 (work at 39, work at 40); age 40 (work at 40, retire at 41); age 41 (retire for the rest of life).

1. Write the FOC and Euler equations for the young from age 1 to 39.
2. Write the FOC and Euler for age 40 only.
3. Write the Euler from age 41 to 60.
4. Write 3 functions that solve the decision rules for young age, age 40, and retirees.
5. Next, write 1 iteration, given $k[61], n[61], k[61], n[60]$, perform backward iteration and solve until $n[1], k[1]$.
6. Next, write a program of many iterations as step 5. For each iteration, you calculate the error value and update new guesses of n, k using the second method.
7. Step 6 completes an inner loop. Compress it into a function, taking the guess of N as argument (input), and output 2 arrays: age profile k^s and n^s.
8. Write an outer loop. In each loop, take N as given, run the backward iteration function written in step 6, output the new aggregate N, and then update the new guess if necessary.

Tips

- Because we are solving backward, it is better to use Julia because Julia's index system starts from 1.
- Technically, once the agent retires, he supplies zero labor and only cares about how much to consume and save, which is governed by the Euler equation. Hence, the current capital holding can be solved explicitly.
- For young agents, you need to solve simultaneously a system of 2 equations (FOC and Euler) of 2 variables (consumption and labor). As the model cannot be solved explicitly, use this code snippet to solve:

```
using NLsolve
# solving for x,y -> z, with parameters a,b,c
function system_eq!(F, z, a, b, c)
  x, y = z
  F[1] = x^2 + y^2 - a + 2b
  F[2] = (x * y)^c - a - b^2
end
#specify values of parameters
a = 1.0
b = 2.0
c = 2.0
z_guess = [0.0, 0.0]  # Initial guess for x and y
# Solve the system of equations
result = nlsolve((F, z) -> system_eq!(F, z, a, b, c), z_guess)
# Extract the solution
x_solution = result.zero[1]
y_solution = result.zero[2]
```

Note: The correct guess is very important when applying this solver. The closer it is to the true solutions, the more accuracy it gets. With backward iteration, I recommend taking the previously solved known value as the guess. For example, in Auerbach-Kotlikoff, the guess to solve k^{39} should take the initial guess (for the solver) of k and n from k^{40}, k^{41}, n^{40}.

- You should start with step 3. Once you tested its reliability, write it into a function. After that, write an outer loop for steps 2 to 5.

More on step 3

For backward iteration, since $k^{61} = 0 = k^1$, we only need to calculate the decision rule for $k^{59}, k^{58}, \ldots, k^2$.

For $t = 59, \ldots, 41$: Since the retiree does not work, we only need to deal with the Euler equation to determine k. To calculate the optimal k^{59}, we need the information of k^{60} and k^{61}. Although k^{61} is known, k^{60} is not. Here, we provide the first 2 guesses for the first 2 iterations

$$k_1^{60} = 0.15, k_2^{60} = 0.2$$

the subscript denotes the i^{th} iteration.

For $t = 40$, you need to find the optimal values of $n^{40}(n^{41}, k^{41})$, $k^{40}(k^{41}, k^{42}, n^{41}, n^{42})$. As this is the last period with a positive labor supply, we have $n^{41} = n^{42} = 0$. Capital holdings k^{41}, k^{42} have been solved in the retiree's problem. Hence, you have 2 equations of 2 unknowns. We can use Julia's 'NLSolve'.

For $t = 39, \ldots, 1$, we will do things similar to the previous steps. Note that when we calculate k^1, if it is different from the tolerance, we must update K and N.

Updating k^{60} using the Secant Method:

$$k_i^{60} = k_{i-1}^{60} - \frac{k_{i-1}^{60} - k_{i-2}^{60}}{k_{i-1}^{1} - k_{i-2}^{1}} k_{i-1}^{1}$$

More on steps 1 and 5

It is actually sufficient to make only a guess of N since K must scale up with it due to Cobb-Douglas technology. Since there is a target for the steady state N, it is a more appropriate approach than guessing randomly. In the code, *nbar* is the aggregate N.

The update algorithm is simple:

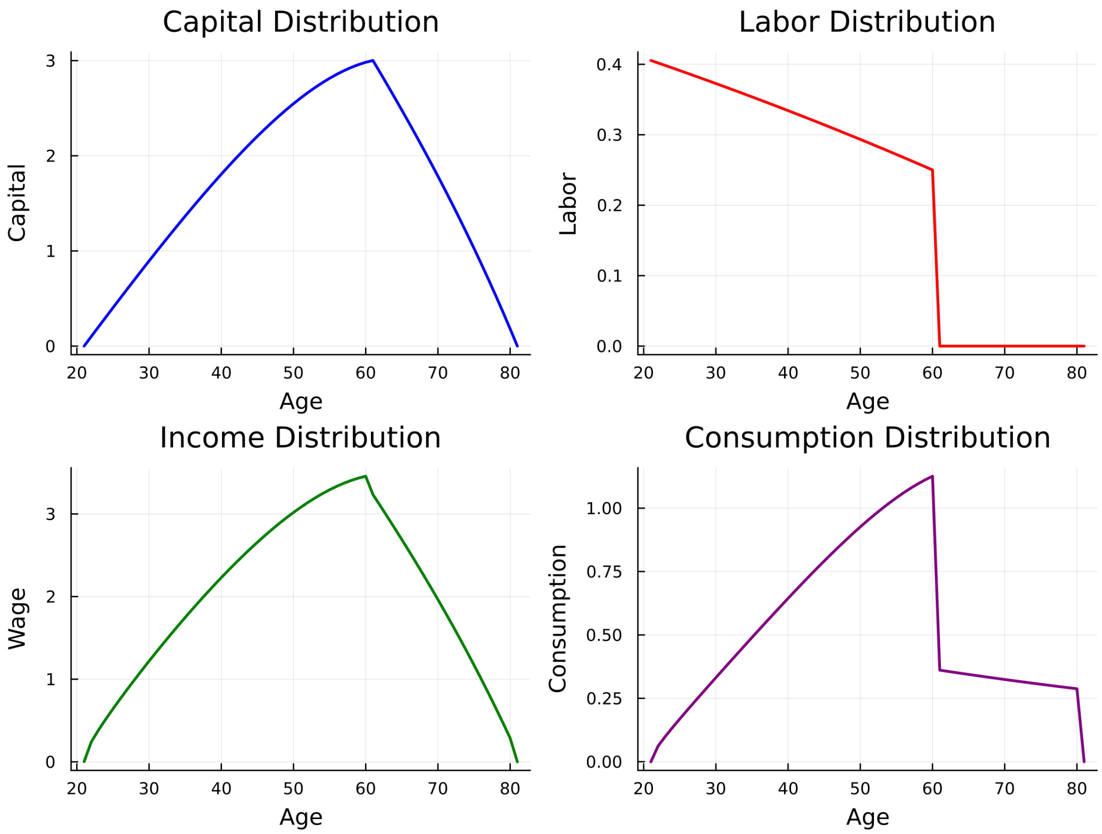
$$N_j^{guess} = \phi N_{j-1}^{guess} + (1-\phi) N_{j-1}^{out}$$

with ϕ is the learning rate, j is the j^{th} iteration. N^{out} is the outcome of an iteration taking N^{guess} as an input. The algorithm should update and converge at iteration $\hat{j}$ such that $N_{\hat{j}}^{guess} = N_{\hat{j}}^{out}$.

For the first 2 initial guesses, use $N_1 = 0.15$, $N_2 = 0.25$.

Results

You should find the steady state of K and N as 0.913 and 0.221, respectively.



Further Applications

Two other solution methods exist Value function iteration and projection methods. As they require more advanced programming, this camp does not have enough time to cover them. As indicated in Heer and Maußner (2009), the direct computation is fast, reliable, and accurate. Whenever possible, this method should be applied. You can extend this model with more sophisticated features, such as

- age-specific productivity (Huggett, 1996)
- age-specific survival probabilities (İmrohoroglu et al., 1995; Huggett and Ventura, 1999)
- money holding (Heer et al., 2007)
- government debt (Braun and Joines, 2015)

One limitation of this method is that it is not feasible to solve the model with idiosyncratic uncertainty, such as stochastic autocorrelated productivity shocks. In such an environment, you need to use value function iteration or projection methods.

References

Acemoglu, D. (2008). Introduction to modern economic growth. Princeton University Press.

Braun, R. A. and Joines, D. H. (2015). The implications of a graying japan for government policy. Journal of Economic Dynamics and Control, 57:1–23.

Costa, C. (2018). Understanding DSGE models: theory and applications. Vernon Press.

Feldstein, M. and Auerbach, A. J. (2002a). Handbook of Public Economics (Volume 3). Elsevier.

Feldstein, M. and Auerbach, A. J. (2002b). Handbook of Public Economics (Volume 4). Elsevier.

Hansen, G. D. and Prescott, E. C. (1995). Recursive methods for computing equilibria of business cycle models. Frontiers of business cycle research. Nova Jersey: Princeton University, pages 39–64.

Heer, B. (2019). Public economics: The macroeconomic perspective.

Heer, B. and Maußner, A. (2005). Dynamic General Equilibrium Modeling: Computational Methods and Applications (1st edition). Springer.

Heer, B. and Maußner, A. (2009). Dynamic General Equilibrium Modeling: Computational Methods and Applications (2nd edition).

Heer, B. and Maußner, A. (2024). Dynamic General Equilibrium Modeling: Computational Methods and Applications (3rd edition). Springer.

Heer, B., Maussner, A., and McNelis, P. D. (2007). The Money-Age Distribution: Empirical Facts and Limited Monetary Models. Technical report.

Huggett, M. (1996). Wealth distribution in life-cycle economies. Journal of Monetary Economics, 38(3):469–494.

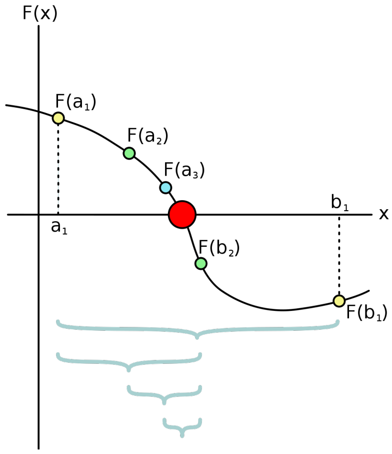
Huggett, M. and Ventura, G. (1999). On the distributional effects of social security reform. Review of Economic Dynamics, 2(3):498–531.

İmrohoroglu, A., Imrohoroglu, S., and Joines, D. H. (1995). A life cycle analysis of social security. Economic theory, 6:83–114.

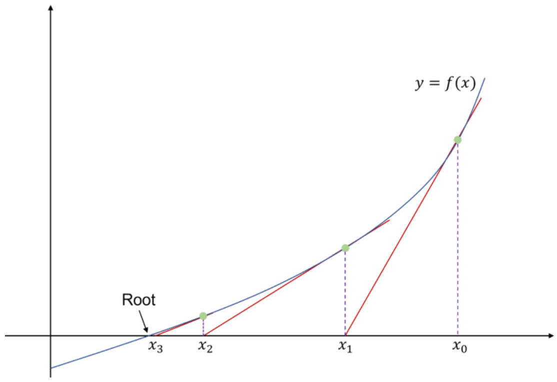
Torres, J. L. (2020). Introduction to dynamic macroeconomic general equilibrium models. Vernon Press.

A Illustrations of Root Finding Algorithms

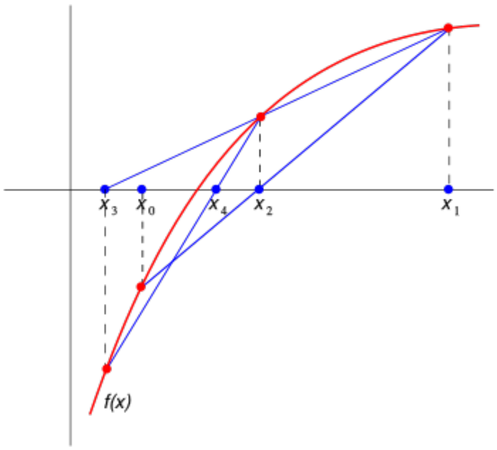
A.1 Bisection



A.2 Newton



A.3 Secant



B Other Iterative Methods

B.1 Euler Equation Iteration

Consider the Ramsey model with log utility, full depreciation and $y_t = Ak_t^\alpha$.

$$\max \sum_{t=0}^{\infty} \beta^t \ln c_t,$$
$$s.t. \;\; c_t + k_{t+1} = Ak_t^\alpha$$

The Euler equation becomes

$$\frac{1}{c_t} = \frac{1}{c_{t+1}} A\alpha\beta k_{t+1}^{\alpha-1} = \alpha\beta \frac{1}{c_{t+1}} \frac{y_{t+1}}{k_{t+1}}$$

The Transversality condition

$$\lim_{t\to\infty} \beta^T u'(c_T) k_{t+T} = \beta^T \frac{k_{t+T}}{c_T} 0.$$

Using the budget constraint

$$c_t + k_{t+1} = y_t$$

and adding 1 to both sides

$$\frac{c_t + k_{t+1}}{c_t} = 1 + \alpha\beta \frac{c_{t+1} + k_{t+2}}{c_{t+1}} \tag{38}$$

Define

$$z_t = \frac{c_t + k_{t+1}}{c_t} \equiv \frac{y_t}{c_t}$$

Iterating (38), we get

$$\begin{aligned} z_t &= 1 + \alpha\beta z_{t+1} = 1 + \alpha\beta + (\alpha\beta)^2 z_{t+2} \\ &= \sum_{t=0}^{\infty} (\alpha\beta)^t + \lim_{T\to\infty} (\alpha\beta)^T z_{t+T} \\ &= \frac{1}{1-\alpha\beta} \quad \text{since } \alpha\beta < 1 \text{ and the transversality condition holds} \end{aligned}$$

If we use the definition of z_t, it is easy to solve

$$c_t = (1-\alpha\beta) y_t,$$
$$k_{t+1} = \alpha\beta y_t.$$

B.2 Policy Function Iteration

This section borrows from https://www.eco.uc3m.es/~jrincon/Teaching/Master/SDDP.pdf#page=8.70. First, pick a feasible policy function

$$k_{t+1} = h_0(k) = 0.5Ak^\alpha$$

The value function is

$$\begin{aligned}
V^{h_0}(k) &= \sum_{t=0}^{\infty} \beta^t \ln(Ak_t^\alpha - 0.5Ak_t^\alpha) \\
&= \sum_{t=0}^{\infty} \beta^t \ln(0.5Ak_t^\alpha) \\
&= \sum_{t=0}^{\infty} \beta^t \left(\ln(0.5A) + \alpha \ln k\right)
\end{aligned}$$

Note that

$$k_t = 0.5Ak_{t-1}^\alpha = 0.5A(0.5Ak_{t-2}^\alpha)^\alpha = 0.5^{\alpha+1}A^{\alpha+1}k_{t-2}^{\alpha^2},$$

implying that

$$k_t = Dk_0^{\alpha^t}$$

Substituting this into V^{h_0} yields

$$V^{h_0}(k) = \sum_{t=0}^{\infty} \beta^t (\ln(0.5A) + \alpha \ln D + \alpha^{t+1} \ln k_0) = E + \frac{\alpha}{1 - \beta\alpha} \ln k_0$$

We then compute

$$\max_{k'} V := \ln(Ak^\alpha - k') + \beta \left(E + \frac{\alpha}{1 - \beta\alpha} \ln k' \right)$$

FOC

$$-\frac{1}{Ak^\alpha - k'} + \frac{\beta\alpha}{A - \beta\alpha} \frac{1}{k'} = 0$$

thus

$$k' = \alpha\beta Ak^\alpha$$

The policy improvement algorithm converges in a single step.

C Codes

C.1 Ramsey's Value Function Iteration

(Python)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize_scalar

# params
beta = 0.98
delta = 0.1
theta = 0.36

# Define global variables
vlast = np.zeros(100)
k0 = np.linspace(0.1, 6, 100)

# Function to calculate the value function
def valfun(k): # k as k'
    global vlast, beta, delta, theta, kt
    g = np.interp(k, k0, vlast)
    c = kt**theta - k + (1 - delta) * kt
    if c <= 0:
        val = -888 - 800 * abs(c)
    else:
        val = np.log(c) + beta * g
    return -val

# Initialize arrays
v = np.zeros(100)
kt1 = np.zeros(100)
numits = 1000
error = 1
tol = 1e-6
its = 0

# Begin recursive calculations
while error > tol and its < numits:
    for j in range(100):
        kt = k0[j]
        ktp1 = minimize_scalar(valfun, bounds=(0.01, 6.2),
        ↪  method='bounded').x
        v[j] = -valfun(ktp1)
        kt1[j] = ktp1

    if its % 48 == 0:
        plt.plot(k0, v, label='iter. ' + str(its))
        plt.xlabel('k(t)', fontsize=16)
        plt.ylabel('V(k(t))', fontsize=16)
        plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
        plt.draw()

    error = np.max(np.abs(v - vlast))
    its += 1
    vlast = v.copy()
```

```
plt.show()

# Plot the policy function
plt.plot(k0, kt1)
plt.plot(k0, k0, ls=':', label='45 degree line')
plt.xlabel('k(t)', fontsize=16)
plt.ylabel('k(t+1)', fontsize=16)
plt.show()
```

C.2 OLG Direct Computation

(in Julia)

```
using Plots
using NLsolve


# ========================
# == Parameters ==========
# ========================
beta = 0.98
gamma = 2
alpha = 0.36
delta = 0.1
eta = 2
repl = 0.3
T = 40
TR = 20
tau = repl / (2 + repl)
psi = 0.001
r = 0.045
phi = 0.5  # Learning rate for nbar update


# key variables calculation
function update_w(k, n)
    return (1 - alpha) * k^alpha * n^(-alpha)
end

function update_kbar(n, r)
    return n * (alpha / (r + delta))^(1 / (1 - alpha))
end

function update_b(w, n)
    return 0.3 * ((1-tau) * w * n)
end

# ==========================================================
# == Decision Solvers =====================================
# == these functions solve the decision rules for ks and ns=
# ==========================================================

# 1. Going to use for s = 59, 58, ..., 41
function solve_ks_old(ks1, ks2)
```

```
    ks = (1 / (1 + r)) * ((beta * (1 + r))^(1 / eta) * ((1 + r) * ks1
    ↪  + b - ks2 + psi) - (b - ks1 + psi))
    return ks
end

# 2. At s = 40, given ks41 and ns41, solve ks40 and ns40
function fs40!(F, X, ks1, ks2)
    ns1 = 0
    ks, ns = X
    F[1] = (1 - tau) * w * (1 - ns) / gamma - ((1 + r) * ks + (1 -
    ↪  tau) * w * ns - ks1 + psi)
    F[2] = 1 / beta - (((1 + r) * ks1 + (1 - tau) * w * ns1 - ks2 +
    ↪  psi) / ((1 + r) * ks + (1 - tau) * w * ns - ks1 + psi))^(-eta)
    ↪  * (((1 - ns1) / (1 - ns))^(gamma / (1 - eta))) * (1 + r)
end

# 3. At s = 39, 38, ... , 1, given ks[s+1], ns[s+1], ks[s+2], solve
↪  ks[s] and ns[s]
function fs_young!(F, X, ks1, ns1, ks2)
    ks, ns = X
    F[1] = (1 - tau) * w * (1 - ns) / gamma - ((1 + r) * ks + (1 -
    ↪  tau) * w * ns - ks1 + psi)
    F[2] = 1 / beta - (((1 + r) * ks1 + (1 - tau) * w * ns1 - ks2 +
    ↪  psi) / ((1 + r) * ks + (1 - tau) * w * ns - ks1 + psi))^(-eta)
    ↪  * (((1 - ns1) / (1 - ns))^(gamma / (1 - eta))) * (1 + r)
end

#
↪  ========================================================================
# == Backward Iteration Function
↪  ======================================
# == input: given nbar, solve the decision rules and ss age-profile
↪  ===
# == output: series of steady state age-profile ks_true and ns_true
↪  ==
# == in the inner loop, first guess k[60] and iterate until k[1] = 0
↪  ==
#
↪  ========================================================================
# Target: loop backward iteration until we get k[1] = 0
# change the guess of k[60] and iterate again if k[1] is not 0
max_iter = 30

function backward_iteration(nbar)
    k60_guess = zeros(max_iter + 1)
    k1_res = zeros(max_iter + 1)

    # true value
    ks_true = zeros(61)
    ns_true = zeros(61)

    # set tol value
    i = 1
    tol = 1e-6
    err = 0.1

    while i <= max_iter && abs(err) > tol
```

```
        # an empty array to store the results
        ks = zeros(61)
        ns = zeros(61)
        # update k60 guess
        if i == 1
            k60_guess[i] = 0.15
        elseif i == 2
            k60_guess[i] = 0.2
        else
            # update by secant method
            k60_guess[i] = k60_guess[i-1] - (k1_res[i-1] - 0) *
            ↪  (k60_guess[i-1] - k60_guess[i-2]) / (k1_res[i-1] -
            ↪  k1_res[i-2])
        end
        # initiate a guess for k[60]
        ks[60] = k60_guess[i]
        # calculate k[s] and n[s] for s = 59, 58, ..., 1
        for s in 59:-1:1
            if s >= 41
                # at s = 59, 58, ..., 41, given k[s+1] and k[s+2],
                ↪  solve k[s]
                ks[s] = solve_ks_old(ks[s+1], ks[s+2])
                ns[s] = 0
            elseif s == 40
                # at s = 40, given k41 and n41, solve k40 and n40
                result = nlsolve((F, X) -> fs40!(F, X, ks[s+1],
                ↪  ks[s+2]), [ks[s+1], ns[s+1]])
                ks[s] = result.zero[1]
                ns[s] = result.zero[2]
            else
                # at s = 39, 38, ..., 1, given k[s+1] and n[s+1],
                ↪  solve k[s] and n[s]
                result = nlsolve((F, X) -> fs_young!(F, X, ks[s+1],
                ↪  ns[s+1], ks[s+2]), [ks[s+1], ns[s+1]])
                ks[s] = result.zero[1]
                ns[s] = result.zero[2]
            end
        end
        # store the results
        ks_true = ks
        ns_true = ns
        # store k1 and calculate the error
        k1_res[i] = ks[1]
        # update error value
        err = ks[1]
        # increase the iteration if the error is still greater than
        ↪  the tolerance, otherwise break the loop
        if abs(err) > tol
            i += 1
        else
            break
        end
    end

    return ks_true, ns_true
end
```

```
#
↪ ==========================================================================
# == Outer Loop Algorithm
↪ ==============================================
# == input: a guess of nbar distribution
↪ ==============================
# == output: the true nbar with its associated ks_true and ns_true
↪ ===
#
↪ ==========================================================================

outer_max_iter = 30
outer_tol = 1e-6
outer_err = 1.0
outer_i = 1

# results
K = 0
N = 0
Ny = 0
ks_true = zeros(61)
ns_true = zeros(61)

# Initial guess for nbar
nbar = 0.2
kbar = update_kbar(nbar, r)
w = update_w(kbar, nbar)
b = update_b(w, nbar)

# the loop algorithm
while outer_i <= outer_max_iter && abs(outer_err) > outer_tol
    kbar = update_kbar(nbar, r)
    w = update_w(kbar, nbar)
    b = update_b(w, nbar)

    # solve the decision rules steady state age-profile
    ks_true, ns_true = backward_iteration(nbar)

    # calculate aggregate capital
    K = sum(ks_true) / (T + TR)

    # calculate aggregate labor
    N = sum(ns_true) / (T + TR)
    Ny = sum(ns_true) / T   #workers only

    # nbar error
    nbar_error = N - nbar
    #println("nbar error: ", nbar_error)

    # update nbar
    if abs(nbar_error) > outer_tol
        nbar_new = phi * nbar + (1 - phi) * N
        nbar = nbar_new
        outer_err = nbar_error
        outer_i += 1
    else
        break
```

```
    end
end

#
↪  =================================================================
# == Calculate other variables
↪  ========================================
# == earnings, consumption, welfare
↪  ===================================
#
↪  =================================================================

# earnings
function cal_income(s, k, n)
    if s <= 40
        return (1 + r) * k + (1 - tau) * n * ((1 - alpha) * k^alpha *
        ↪  n^(-alpha))
    elseif s <= 60
        return (1 + r) * k + b
    else
        return 0
    end
end

# consumption
function cal_c(s, n, w, k1)
    if s <= 40
        return (1 - tau) * w * (1-n) / gamma - psi
    elseif s <= 60
        return w - k1
    else
        return 0
    end
end

# welfare
function cal_welfare(s, c, n)
    return beta^(s - 1) * (((c + psi) * ((1 - n)^gamma))^(1 - eta) -
    ↪  1) / (1 - eta)
end

# using ks_true and ns_true to calculate the variables
ws_true = zeros(61)
cs_true = zeros(61)
us_true = zeros(61)
age_true = zeros(61)

for s in 1:60
    age_true[s] = s + 20
    ws_true[s] = cal_income(s, ks_true[s], ns_true[s])
    cs_true[s] = cal_c(s, ns_true[s], ws_true[s], ks_true[s+1])
    us_true[s] = cal_welfare(s, cs_true[s], ns_true[s])
end

ws_true[61] = 0
cs_true[61] = 0
us_true[61] = 0
```

```
age_true[61] = 81

# ========================================
# == Print the steady states ============
# ========================================

println("Final nbar: ", nbar)
println("Final K: ", kbar)
println("Final N: ", N)
println("Final Ny: ", Ny)
println("b: ", b)

# ========================
# == Plotting ============
# ========================
# show 4 plots at the same time

# plot of ks

plotk = plot(age_true, ks_true, label="ks", title="Capital
↪  Distribution", xlabel="Age", ylabel="Capital", legend=false,
↪  color=:blue, lw=2, dpi=300)
plotn = plot(age_true, ns_true, label="ns", title="Labor
↪  Distribution", xlabel="Age", ylabel="Labor", legend=false,
↪  color=:red, lw=2, dpi=300)
plotw = plot(age_true, ws_true, label="ws", title="Income
↪  Distribution", xlabel="Age", ylabel="Wage", legend=false,
↪  color=:green, lw=2, dpi=300)
plotc = plot(age_true, cs_true, label="cs", title="Consumption
↪  Distribution", xlabel="Age", ylabel="Consumption", legend=false,
↪  color=:purple, lw=2, dpi=300)

fig = plot(plotk, plotn, plotw, plotc, layout=(2, 2), legend=false,
↪  size=(800, 600))

#savefig(fig, "AK60.png")
#savefig(plotw, "AK60w.png")
```