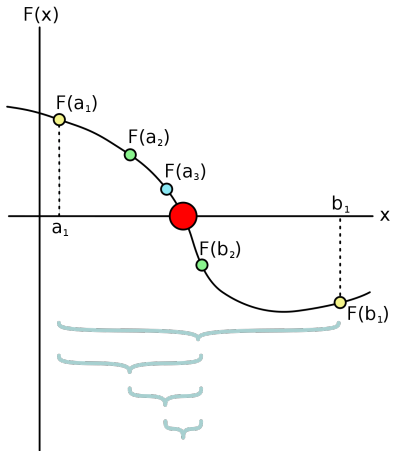
A Excercise List

Ex	Page	Content	Hint
1	5	Prove the existence of the fixed point	show that the LHS is increasing, bounded from 0 to $+\infty$, while the RHS is decreasing, bounded from ∞ to 0. Thus, they must cut somewhere between $[0, \infty)$.
2	7	write 3 programs to find root	
3	9	write programs to find root of a CIES	rewrite it like Eq.(12)
4	12	Solve for Ramsey's steady states	derive $u'(c)$, use Euler to solve for $\bar{k}$, then use the resource constraint to derive $\bar{c}$.
5	12	write a program for undetermined coeff.	find $\bar{k}$, create an array of k from 0 to $1.5 \times \bar{k}$. Write a function `policy_func(k)` and apply the function on the array k.
6	16	write a program of the perturbation method for policy function	similar to the previous, write a policy function and apply it on the array of k.
7	18	value function iteration	see Appendix C.1
8	20	solve for the FOC of the OLG	just follow the steps
9	23	solve the OLG by direct computation	do it incrementally. Check this Github link for each step. 1. `agentopt.jl` is step 4, which solves the first 3 steps. 2. `agentopt_1iter.jl` completes step 6. 3. `agentopt_func.jl` writes the whole inner loop into a function (step 7). 4. `agentopt_multi_iter.jl` completes step 8, where we loop with each guess of initial N.

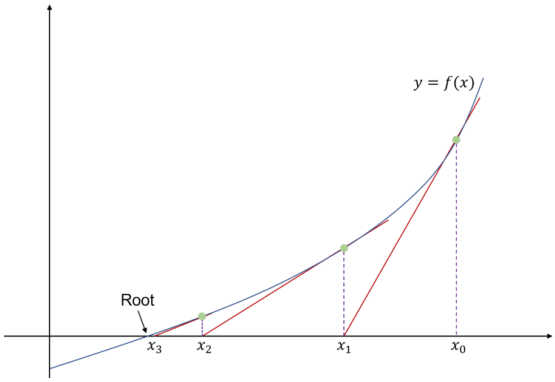
B Appendix

B.1 Illustrations of Root Finding Algorithms

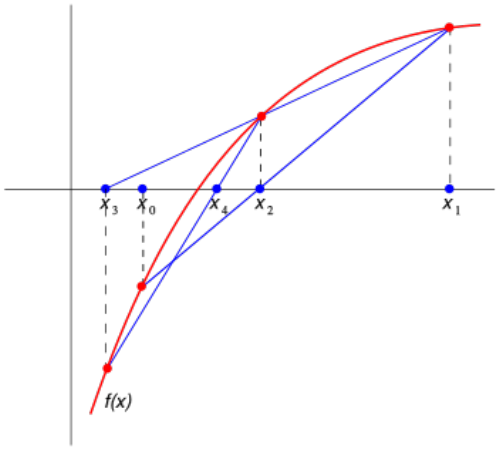
Bisection



Newton-Raphson



Secant



B.2 Other Iterative Methods to Solve Ramsey

B.2.1 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

$$\frac{c_t + k_{t+1}}{c_t} = 1 + \alpha\beta \frac{c_{t+1} + k_{t+2}}{c_{t+1}}. \tag{39}$$

Define

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

Iterating (39), 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.2.2 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

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

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

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

The 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

(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")
```