Macroeconomics II Homework 2

Graduate School of Economics, The University of Tokyo

29–246029 Rin NITTA
29-246033 Rei HANARI
29–246004 Kosuke IGARASHI

November 4, 2024

1

(b)

The value function and the policy function are shown in the following figures.

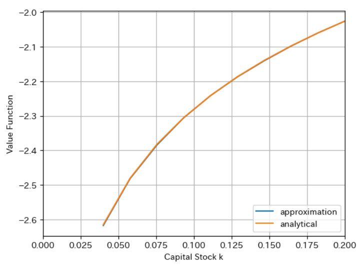

Figure1 Value Function

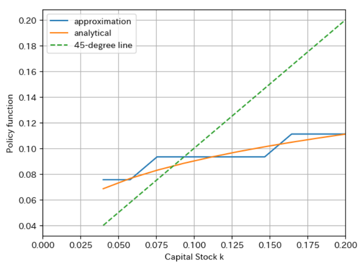

Figure2 Policy Function

The Python code is as follows:

```
import numpy as np
import matplotlib.pyplot as plt

class Model():
    def __init__(self,
```

```
        beta = 0.6,
        gamma = 1.0,
        alpha = 0.3,
        delta = 1.0,
        nk = 10,
        kmax = 0.2,
        kmin = 0.04,
        maxit = 1000,
        tol = 1e-4,
        ):

        self.beta, self.gamma, self.alpha = beta, gamma, alpha
        self.delta, self.nk = delta, nk
        self.kmax, self.kmin = kmax, kmin
        self.maxit, self.tol = maxit, tol
        self.kgrid = np.linspace(kmin,kmax,nk)


it = 1
dif1 = 1.0
dif2 = 1.0

m = Model()
vfcn = np.ones(m.nk)
pfcn = np.ones_like(vfcn)
Tvfcn = np.zeros_like(vfcn)
Tpfcn = np.zeros_like(vfcn)
vkp = np.empty((m.nk,m.nk))
v_conv = []
p_conv = []
util = np.ones((m.nk, m.nk))

def utility(c):
    if c > 0:
        return np.log(c)
    else:
        return -1e10

for i in range(m.nk):
    for j in range(m.nk):
        wealth = m.kgrid[i] ** m.alpha + (1.0 - m.delta) * m.kgrid[i]
        cons = wealth - m.kgrid[j]
        util[i, j] = utility(cons)

while (it<m.maxit) & (dif1>m.tol):

    for i in range(m.nk):

        vkp[i,:] = util[i,:] + m.beta*vfcn

        ploc = np.argmax(vkp[i,:])
        Tvfcn[i] = vkp[i,ploc]
        Tpfcn[i] = m.kgrid[ploc]

    dif1 = np.max(np.abs((Tvfcn-vfcn)/vfcn))
    dif2 = np.max(np.abs((Tpfcn-pfcn)/pfcn))
```

```python
    vfcn = np.copy(Tvfcn)
    pfcn = np.copy(Tpfcn)

    print(f"iteration index: {it}, iteration diff of value: {dif1:.7f}")

    v_conv.append(dif1)
    p_conv.append(dif2)

    it += 1

print("-+- PARAMETER VALUES -+-")
print(f"beta={m.beta}, gamma={m.gamma}, alpha={m.alpha}, delta={m.delta}")
print(f"kmin={m.kmin}, kmax={m.kmax}, grid={m.nk}")

AA = (1-m.beta)**(-1) * (np.log(1-m.alpha*m.beta) + ((m.alpha*m.beta)/(1-m.alpha*m.beta))*np.
    log(m.alpha*m.beta))
BB = m.alpha/(1-m.alpha*m.beta)
v_true = AA + BB*np.log(m.kgrid)
p_true = m.beta * m.alpha * (m.kgrid ** m.alpha)

fig, ax = plt.subplots()
ax.plot(m.kgrid,vfcn,label="approximation")
ax.plot(m.kgrid,v_true,label="analytical")
ax.set(title="",xlabel=r"Capital Stock k", ylabel=r"Value Function",xlim=(0,m.kmax))
ax.legend(loc="lower right")
ax.grid()
plt.show()

fig, ax = plt.subplots()
ax.plot(m.kgrid, pfcn, label="approximation")
ax.plot(m.kgrid, p_true, label="analytical")
ax.plot(m.kgrid, m.kgrid, ls="--", label="45-degree line")
ax.set(title="",xlabel=r"Capital Stock k", ylabel=r"Policy function",xlim=(0,m.kmax))
ax.legend(loc="upper left")
ax.grid()
plt.show()
```

(c)

The value function and the policy function are shown in the following figures.

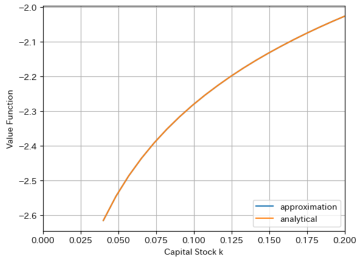

Figure3 Value Function

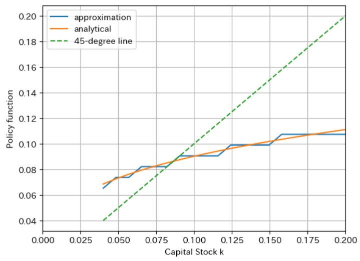

Figure4 Policy Function

The Python code is as follows:

```
import numpy as np
import matplotlib.pyplot as plt

class Model():
    def __init__(self,
        beta = 0.6,
        gamma = 1.0,
        alpha = 0.3,
        delta = 1.0,
        nk = 20,
        kmax = 0.2,
        kmin = 0.04,
        maxit = 1000,
        tol = 1e-4,
        ):

        self.beta, self.gamma, self.alpha = beta, gamma, alpha
        self.delta, self.nk = delta, nk
        self.kmax, self.kmin = kmax, kmin
        self.maxit, self.tol = maxit, tol
        self.kgrid = np.linspace(kmin,kmax,nk)

it = 1
dif1 = 1.0 #価値関数の繰り返し誤差
dif2 = 1.0 #政策関数の繰り返し誤差

m = Model()
vfcn = np.ones(m.nk)
pfcn = np.ones_like(vfcn)
Tvfcn = np.zeros_like(vfcn)
Tpfcn = np.zeros_like(vfcn)
```

```python
vkp = np.empty((m.nk,m.nk))
v_conv = []
p_conv = []

util = np.ones((m.nk, m.nk))

def utility(c):
    if c > 0:
        return np.log(c)
    else:
        return -1e10

for i in range(m.nk):
    for j in range(m.nk):
        wealth = m.kgrid[i] ** m.alpha + (1.0 - m.delta) * m.kgrid[i]
        cons = wealth - m.kgrid[j]
        util[i, j] = utility(cons)

while (it<m.maxit) & (dif1>m.tol):

    for i in range(m.nk):

        vkp[i,:] = util[i,:] + m.beta*vfcn

        ploc = np.argmax(vkp[i,:])
        Tvfcn[i] = vkp[i,ploc]
        Tpfcn[i] = m.kgrid[ploc]

    dif1 = np.max(np.abs((Tvfcn-vfcn)/vfcn))
    dif2 = np.max(np.abs((Tpfcn-pfcn)/pfcn))

    vfcn = np.copy(Tvfcn)
    pfcn = np.copy(Tpfcn)

    print(f"iteration index: {it}, iteration diff of value: {dif1:.7f}")

    v_conv.append(dif1)
    p_conv.append(dif2)

    it += 1

print("-+- PARAMETER VALUES -+-")
print(f"beta={m.beta}, gamma={m.gamma}, alpha={m.alpha}, delta={m.delta}")
print(f"kmin={m.kmin}, kmax={m.kmax}, grid={m.nk}")

AA = (1-m.beta)**(-1) * (np.log(1-m.alpha*m.beta) + ((m.alpha*m.beta)/(1-m.alpha*m.beta))*np.
    log(m.alpha*m.beta))
BB = m.alpha/(1-m.alpha*m.beta)
v_true = AA + BB*np.log(m.kgrid)
p_true = m.beta * m.alpha * (m.kgrid ** m.alpha)


fig, ax = plt.subplots()
ax.plot(m.kgrid,vfcn,label="approximation")
ax.plot(m.kgrid,v_true,label="analytical")
ax.set(title="",xlabel=r"Capital Stock k", ylabel=r"Value Function",xlim=(0,m.kmax))
ax.legend(loc="lower right")
```

```
ax.grid()
plt.show()

fig, ax = plt.subplots()
ax.plot(m.kgrid, pfcn, label="approximation")
ax.plot(m.kgrid, p_true, label="analytical")
ax.plot(m.kgrid, m.kgrid, ls="--", label="45-degree line")
ax.set(title="",xlabel=r"Capital Stock k", ylabel=r"Policy function",xlim=(0,m.kmax))
ax.legend(loc="upper left")
ax.grid()
plt.show()
```

2

(a)

At the optimal consumption, the budget constraint holds with equality:

$$c_t = Ak_t^\alpha - k_{t+1}$$

Define

$$w(k_0) = \max_{\{k_{t+1}\}_{t=0}^\infty} \sum_{t=0}^\infty \beta^t \log c_t$$

Then, this is equivalent to

$$\begin{aligned} w(k_0) &= \max_{\{k_{t+1}\}_{t=0}^\infty ; k_0 \text{ given}} \sum_{t=0}^\infty \beta^t \log c_t \\ &= \max_{k_1 ; k_0 \text{ given}} \log c_0 + \beta \left[\max_{\{k_{t+1}\}_{t=1}^\infty ; k_1 \text{ given}} \sum_{t=1}^\infty \beta^{t-1} \log c_t \right] \\ &= \max_{k_1 ; k_0 \text{ given}} \log(Ak_0^\alpha - k_1) + \beta w(k_1) \end{aligned}$$

with conditions $0 \le k_1 \le Ak_0^\alpha$ for all $t = 0, \cdots, \infty$. This is the recursive problem of the social planner.

(b)

(i): Guess and Verify

Guess $V(k) = X + Y \log Ak^\alpha$ where X and Y are coefficients to be determined. Then, the Bellman equation is

$$V(k) = \max_{0 \le k' \le Ak^\alpha ; k \text{ given}} \log(Ak^\alpha - k') + \beta(X + Y \log Ak'^\alpha)$$

The first-order condition is

$$\begin{aligned} \frac{\partial V(k)}{\partial k'} &= -\frac{1}{Ak^\alpha - k'} + \alpha\beta Y \frac{1}{k'} = 0 \\ \therefore \quad k' &= \frac{\alpha\beta Y}{1 + \alpha\beta Y} Ak^\alpha \end{aligned}$$

Evaluating the objecting function (RHS) at the optimal k', we have

$$
\begin{aligned}
(RHS) &= \log\left(Ak^\alpha - \frac{\alpha\beta Y}{1+\alpha\beta Y}Ak^\alpha\right) + \beta(X + Y\log A\left(\frac{\alpha\beta Y}{1+\alpha\beta Y}Ak^\alpha\right)^\alpha) \\
&= \beta X + \log\left(\frac{1}{1+\alpha\beta Y}\right) + \alpha\beta Y\log\left(\frac{\alpha\beta Y}{1+\alpha\beta Y}\right) + \beta Y\log A + (1+\alpha\beta Y)\log(Ak^\alpha)
\end{aligned}
$$

Solving this, we have

$$
\begin{aligned}
X &= \frac{1}{1-\beta}\left[\log(1-\alpha\beta) + \frac{\alpha\beta}{1-\alpha\beta}\log(\alpha\beta)\right] \\
Y &= \frac{1}{1-\alpha\beta}
\end{aligned}
$$

Thus, the value function is represented as the form of $V(k) = C + \frac{1}{1-\alpha\beta}\log Ak^\alpha$.

(ii): Value Function Iteration

We set the value of parameters as follows:

$$
\begin{gathered}
\beta = 0.6, \quad \alpha = 0.3, \quad \delta = 1.0, \quad A = 1.0, \\
nk = 100, \quad k_{\max} = 0.2, \quad k_{\min} = 0.04, \quad \text{maxit} = 10000, \quad \text{tol} = 1e-5
\end{gathered}
$$

The value function and the policy function are shown in the following figures. We used the result of Guess and Verify as the true value function and policy function.

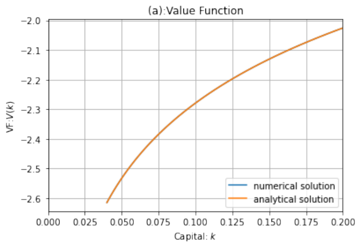

Figure5 Value Function

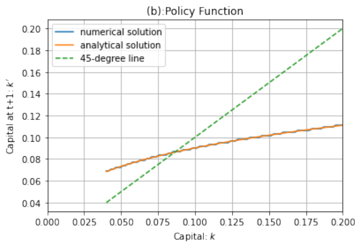

Figure6 Policy Function

The Python code is as follows:

```
class Model():
    def __init__(self,
        beta = 0.6,
```

```python
        alpha = 0.3,
        delta = 1.0,
        A = 1.0,
        nk = 100,
        kmax = 0.2,
        kmin = 0.04,
        maxit = 10000,
        tol = 1e-5,
        ):

        self.beta, self.alpha = beta, alpha
        self.delta, self.nk = delta, nk
        self.A = A
        self.kmax, self.kmin = kmax, kmin
        self.maxit, self.tol = maxit, tol
        self.kgrid = np.linspace(kmin,kmax,nk)


it = 1
dif1 = 1.0
dif2 = 1.0

m = Model()
vfcn = np.ones(m.nk)
pfcn = np.ones_like(vfcn)
Tvfcn = np.zeros_like(vfcn)
Tpfcn = np.zeros_like(vfcn)
vkp = np.empty((m.nk,m.nk))
v_conv = []
p_conv = []
util = np.ones((m.nk,m.nk))

for i in range(m.nk): #k
    for j in range(m.nk): #k'

        wealth = m.A * m.kgrid[i] ** m.alpha + (1.0-m.delta)*m.kgrid[i] #A k^\alpha
        cons = wealth - m.kgrid[j] # A k^\alpha - k'
        util[i,j] = np.log(cons) # log(A k^\alpha - k')

while (it<m.maxit) & (dif1>m.tol):

    for i in range(m.nk):

        vkp[i,:] = util[i,:] + m.beta*vfcn

        ploc = np.argmax(vkp[i,:])
        Tvfcn[i] = vkp[i,ploc]
        Tpfcn[i] = m.kgrid[ploc]

    dif1 = np.max(np.abs((Tvfcn-vfcn)/vfcn))
    dif2 = np.max(np.abs((Tpfcn-pfcn)/pfcn))

    vfcn = np.copy(Tvfcn)
    pfcn = np.copy(Tpfcn)

    print(f"iteration index: {it}, iteration diff of value: {dif1:.7f}")
```

```
    v_conv.append(dif1)
    p_conv.append(dif2)

    it += 1

print("-+- PARAMETER VALUES -+-")
print(f"beta={m.beta} alpha={m.alpha}, delta={m.delta}")
print(f"kmin={m.kmin}, kmax={m.kmax}, grid={m.nk}")

# True value function
AA = (1-m.beta)**(-1) * (np.log(1-m.alpha*m.beta) + ((m.alpha*m.beta)/(1-m.alpha*m.beta))*np.
    log(m.alpha*m.beta))
BB = m.alpha/(1-m.alpha*m.beta)
v_true = AA + BB*np.log(m.kgrid)
p_true = m.beta * m.alpha * (m.kgrid ** m.alpha)

fig, ax = plt.subplots()
ax.plot(m.kgrid,vfcn,label="numerical solution")
ax.plot(m.kgrid,v_true,label="analytical solution")
ax.set(title="(a):Value Function",xlabel=r"Capital: $k$", ylabel=r"VF:$V(k)$",xlim=(0,m.kmax))
ax.legend(loc="lower right")
ax.grid()
plt.show()

fig, ax = plt.subplots()
ax.plot(m.kgrid, pfcn, label="numerical solution")
ax.plot(m.kgrid, p_true, label="analytical solution")
ax.plot(m.kgrid, m.kgrid, ls="--", label="45-degree line")
ax.set(title="(b):Policy Function",xlabel=r"Capital: $k$", ylabel=r"Capital at t+1: $k'$",xlim
    =(0,m.kmax))
ax.legend(loc="upper left")
ax.grid()
plt.show()
```

(iii): Policy Function Iteration

Using the result in (i), we have the true policy function:

$$k' = \frac{\alpha\beta Y}{1+\alpha\beta Y} A k^{\alpha} = \alpha\beta A k^{\alpha}$$

Using this policy function, we can caluculate the value function.

$$\begin{aligned}
V_{h_j}(k_0) &= \sum_{t=0}^{\infty} \beta^t \log\left(A k_t^{\alpha} - \beta A k^{\alpha}\right) \\
&= \sum_{t=0}^{\infty} \beta^t \log k_t^{\alpha} + \frac{1}{1-\beta}\log A + \frac{1}{1-\beta}\log(1-\alpha\beta) \\
&= \frac{\alpha}{1-\alpha\beta}\log k_0 + \frac{1-\beta+\alpha\beta}{(1-\beta)^2}\log((1-\alpha\beta)A)
\end{aligned}$$

Let D be the terms not depending on k_0 in the right hand side of $V_{h_j}(k_0)$. Then,

$$V_{h_j}(k_0) = \frac{\alpha}{1-\alpha\beta}\log k_0 + D$$

From now on, find the k' that maximizes

$$\begin{aligned}\log(Ak^\alpha - k') + \beta V_{h_j}(k') &= \log(Ak^\alpha - k') + \frac{\alpha\beta}{1-\alpha\beta}\log k' + \beta D \\ &= \log\left((Ak^\alpha - k')k'^{\frac{\alpha\beta}{1-\alpha\beta}}\right) + \beta D\end{aligned}$$

The first order condition is

$$\begin{aligned}-k'^{\frac{\alpha\beta}{1-\alpha\beta}} + \frac{\alpha\beta}{1-\alpha\beta}(Ak^\alpha - k')k'^{\frac{\alpha\beta}{1-\alpha\beta}-1} &= 0 \\ \therefore \quad k' = h_{j+1}(k) &= \alpha\beta A k^\alpha\end{aligned}$$

Now, $h_{j+1}(k) = h_j(k) = \alpha\beta Ak^\alpha$, so the the policy function converges.

Thus, we have

$$\begin{aligned}V(k) &= \frac{\alpha}{1-\alpha\beta}\log k + \frac{1}{1-\alpha\beta}\log A + \text{constant} \\ &= \frac{1}{1-\alpha\beta}\log(Ak^\alpha) + \text{constant}\end{aligned}$$

3

(a)

The agent's life-time budget constraint is

$$\sum_{t=0}^{\infty} p_t\left(c_t + \frac{b_{t+1}}{1+r_{t+1}} - b_t - e\right)$$

(b)

Define the Lagrangian as

$$\mathcal{L} = \sum_{t=0}^{\infty}\beta^t\left[\log c_t + \lambda_t\left(p_t\left(c_t + \frac{b_{t+1}}{1+r_{t+1}} - b_t - e\right)\right)\right]$$

The first-order conditions are

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial c_t} &= U'(c_t) - \lambda_t p_t = 0 \\ \frac{\partial\mathcal{L}}{\partial b_t + 1} &= -\beta^t\frac{\lambda_t p_t}{1+r_{t+1}} + \beta^{t+1}\lambda_{t+1}p_{t+1} = 0\end{aligned}$$

Thus, the Euler equation is

$$U'(c_t) = \beta U'(c_{t+1})(1+r_{t+1}) \tag{EE}$$

The transversality condition is

$$\lim_{t\to\infty}\beta^t(1+r_{t+1})U'(c_t)b_t = 0 \tag{TVC}$$

with $c_t = b_t - \frac{b_{t+1}}{1+r_{t+1}} + e$.

(c)

Suppose (EE) and (TVC) hold. Iterating (EE) backward, we have

$$\begin{aligned}
U'(c_t) &= \frac{1}{\beta(1+r_t)} U'(c_{t-1}) \\
&= \frac{1}{\beta(1+r_t)} \frac{1}{\beta(1+r_{t-1})} U'(c_{t-2}) \\
&= \cdots \\
&= \frac{1}{\beta^t} U'(c_0) \prod_{\tau=1}^{t} \frac{1}{1+r_\tau}
\end{aligned}$$

Thus, we have

$$\prod_{\tau=1}^{t} \frac{1}{1+r_\tau} = \beta^t \frac{U'(c_t)}{U'(c_0)}$$

Substituting this into the left hand side of (NPG), we have

$$(LHS) = \frac{1}{U'(c_0)} \lim_{t\to\infty} \beta^t U'(c_t) b_t$$

As for (TVC), it hods that

$$|\beta^t (1+r_{t+1}) U'(c_t) b_t| \geq |\beta^t U'(c_t) b_t| \geq 0$$

for all t, since $r_{t+1} > 0$. Because $\lim_{t\to\infty} |\beta^t (1+r_{t+1}) U'(c_t) b_t| = 0$ by (TVC), then $\lim_{t\to\infty} |\beta^t U'(c_t) b_t| = 0$ also holds. Therefore, $\lim_{t\to\infty} \beta^t U'(c_t) b_t = 0$. Here, (NPG) condition is met, especially with equality.

4

(a)

$$\begin{aligned}
v(k, K) = \max_{c,k\geq 0} U(c) + v(k', K') \\
s.t.\ c + k' &= w(K) + (1 + r(K) - \delta - \tau)k + T \\
K' &= H(K)
\end{aligned}$$

(b)

RCE is a value function $v : \mathbb{R}_+^2 \to \mathbb{R}$ and policy function $C, G : \mathbb{R}_+^2 \to \mathbb{R}_+$ for the representative household, pricing function $w, r : \mathbb{R} \to \mathbb{R}_+$ and an aggregate law of motion $H : \mathbb{R}_+ \to \mathbb{R}_+$ such that

1 Given w, r, H, v solves the Bellman equation and C, G are the associated policy function.
2 Pricing function satisfies firm's FOC.
3 Consistency for all $K \in \mathbb{R}_+$: $H(K) = G(K, K)$

4 Market clearing: for all $K \in \mathbb{R}_+$

$$C(K,K) + G(K,K) = F(K,1) + (1-\delta)K$$

.

(c)

From market clear,

$$\begin{aligned} k_t^d &= k_t^s \\ c_t + k_{t+1} &= f(k_t). \end{aligned}$$

Thus law of motion for aggregate capital stock is

$$K' = H(K) = f(K) - C(K,K).$$

(d)

In the steady state, $K_{t+1} = K_t = K^*$. Thus $I = \delta K^*$ From market clear

$$\begin{aligned} I = \delta K^* &= F(K^*,1) - C(K^*,K^*) \\ K^* &= \frac{1}{\delta}\{F(K^*,1) - C(K^*,K^*)\} \end{aligned}$$

(e)

Law of motion for aggregate variables coincide with CE allocation for representative household and firm.

In CE, Firm solves

$$\max_{\{c_t,k_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} p_t\big(F(K_t^d,1) - w_t - r_t k_t^d\big)$$

From FOC and assumption of F(·),

$$\begin{aligned} F_K(k_t^d,1) &= r_t \\ F_N(k_t^d,1) &= w_t \end{aligned}$$

$$\max_{\{c_t,k_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} p_t\big(F(K_t^d,1) - w_t - r_t k_t^d\big) = 0.$$

Representative household solves

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

$$s.t.$$

$$\sum_{t=0}^{\infty} P_t(c_t + i_t) \leq \sum_{t=0}^{\infty} P_t(w_t + k_t r_t) . c_t, k_{t+1} \geq 0 c_t + i_t = k_{t+1} + (1-\delta)k_t$$

Note that we do not have to consider the tax because government pay back all taxes.

Define Lagrangian as

$$\mathcal{L} = \sum_{t=0}^{\infty} U(c_t) + \mu \left[\sum_{t=0}^{\infty} P_t(w_t + k_t r_t) - \sum_{t=0}^{\infty} P_t(c_t + i_t) \right].$$

From FOCs

$$\begin{aligned} \beta^t U'(c_t) &= \mu P_t \\ P_{t+1} r_{t+1} - P_t + (1-\delta) P_{t+1} &= 0. \end{aligned}$$

Combining, we have Euler equation

$$U'(c_t) = \beta U'(c_{t+1})(1 + r_{t+1} - \delta).$$

From $F(K, N) = AK^\alpha L^{1-\alpha}$, and firm's FOC $F_K(K, 1) = r_t$

$$\begin{aligned} U'(C(K^*, K^*)) &= \beta U'(C(K^*, K^*))(1 + F_K^*(K^*, 1) - \delta) \\ 1 &= \beta(1 + A\alpha(K^*)^{\alpha-1} - \delta) \\ K^* &= \left\{ \frac{1 - \beta(1-\delta)}{\alpha A} \right\}^{\frac{1}{\alpha-1}} \end{aligned}$$

5

The aggregate resource constraint is

$$(1+n)^t c_t + K_{t+1} = F(K_t, (1+n)^t (1+g)^t) + (1-\delta) K_t$$

Define growth-adjusted per-capita variables as

$$\begin{aligned} \tilde{c}_t &= \frac{c_t}{(1+n)^t} \\ \tilde{k}_t &= \frac{k_t}{(1+n)^t} = \frac{k_t}{(1+n)^t (1+g)^t} \end{aligned}$$

Divide the aggregate resource constraint by L_t, we have

$$\begin{aligned} \tilde{c}_t + (1+n)(1+g)\tilde{k}_{t+1} &= f(\tilde{k}_t), \\ \text{where} \quad f(\tilde{k}_t) &= F(\tilde{k}_t, 1) + (1-\delta)\tilde{k}_t. \end{aligned}$$

Rewrite the objective function as

$$\begin{aligned} \sum_{t=0}^{\infty} \beta^t \frac{c_t^{1-\sigma}}{1-\sigma} &= \sum_{t=0}^{\infty} \tilde{\beta}^t \frac{\tilde{c}_t^{1-\sigma}}{1-\sigma}, \\ \text{where} \quad \tilde{\beta} &= \beta(1+n)^{1-\sigma} < 1. \end{aligned}$$

Then, the social planner's problem becomes

$$\max_{\{\tilde{c}_t, \tilde{k}_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \tilde{\beta}^t \frac{\tilde{c}_t^{1-\sigma}}{1-\sigma}$$
$$\text{s.t.} \quad \tilde{c}_t + (1+n)(1+g)\tilde{k}_{t+1} = f(\tilde{k}_t).$$

The Lagrangian is

$$\mathcal{L} = \sum_{t=0}^{\infty} \tilde{\beta}^t \left[\frac{\tilde{c}_t^{1-\sigma}}{1-\sigma} + \sum_{t=0}^{\infty} \lambda_t f(\tilde{k}_t) - \tilde{c}_t - (1+n)(1+g)\tilde{k}_{t+1} \right].$$

The first-order conditions are

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \tilde{c}_t} &= \tilde{\beta}^t(\tilde{c}_t^{-\sigma} - \lambda_t) = 0, \\
\frac{\partial \mathcal{L}}{\partial \tilde{k}_{t+1}} &= -\tilde{\beta}^t \lambda_t (1+n)(1+g) + \tilde{\beta}^{t+1} \lambda_{t+1} \frac{\partial f(\tilde{k}_{t+1})}{\partial \tilde{k}_{t+1}} \\
&= -\tilde{\beta}^t \lambda_t (1+n)(1+g) + \tilde{\beta}^{t+1} \lambda_{t+1} [F'(\tilde{k}_{t+1}, 1) + 1 - \delta] \\
&= 0
\end{aligned}$$

From these equations, we can derive the Euler equation as follows:

$$\tilde{c}_t^{-\sigma} = \frac{\tilde{\beta}}{(1+n)(1+g)} [F'(\tilde{k}_{t+1}, 1) + 1 - \delta] \tilde{c}_{t+1}^{-\sigma}$$

6

For any $\{k_{t+1}\}_{t=0}^{\infty}$, denote log difference as $\dot{x}_{t+1} = \log(x_{t+1}) - \log(x_t)$.
Under BGP, for all t, $\dot{y}_t = \dot{k}_t = 3$

$$\begin{aligned}
\dot{y}_t &= \alpha \dot{k}_t + (1-\alpha)(\dot{z}_t + \dot{l}_t) \\
3 &= 0.4 * 0.3 + (1 - 0.4)(\dot{z}_t + \dot{l}_t) \\
\dot{z}_t &= 2.
\end{aligned}$$

Thus growth rate of TFP along the balanced growth path is 2%.