ECON 6140

Problem Set 1

Gabe Sekeres

February 5, 2025

Skill-biased technical change

1. The elasticity of substitution between capital k_t and skilled labor n_s is defined by:
$$\frac{d\ln(k_t/n_s)}{d\ln(\text{MRS}_{sk})}$$
where the marginal rate of substitution is
$$\text{MRS}_{sk} = \frac{f_{n_s}(z,k_t,n_t)}{f_{k_t}(z,k_t,n_t)} = \frac{\mu n_s^{\rho-1}}{(1-\mu)k_t^{\rho-1}} = \frac{\mu}{1-\mu}\frac{k_t^{1-\rho}}{n_s^{1-\rho}}$$
which means that
$$\ln(\text{MRS}_{sk}) = \ln\left(\frac{\mu}{1-\mu}\right) + (1-\rho)\ln\left(\frac{k_t}{n_s}\right)$$
so we have that
$$\frac{d\ln(\text{MRS}_{sk})}{d\ln(k_t/n_s)} = 1-\rho \Longrightarrow \frac{d\ln(k_t/n_s)}{d\ln(\text{MRS}_{sk})} = \frac{1}{1-\rho}$$
The elasticity of substitution is constant, and dependent only on the parameter ρ.

2. Recall that a steady state must satisfy the first order necessary conditions in the model for it to be interior (rather than the trivial steady state of 0 consumption). Our Lagrangian is
$$\mathcal{L} = \sum_{t=0}^{\infty}\beta^t\left\{u(c_t,1-n_t) + \lambda_t\left[f(z,k_t,n_t) - c_t - x_{k_t}\right] + \theta_t\left[(1-\delta_k)k_t + x_{k_t} - k_{t+1}\right] + \gamma_{1t}c_t + \gamma_{2t}n_t + \gamma_{3t}k_{t+1}\right\}$$
(along with non-negativity constraints and complementary slackness conditions) our first order conditions are
$$\begin{aligned}
0 &= u_c(c_t,1-n_t) - \lambda_t + \gamma_{1t} && (c_t)\\
0 &= -u_n(c_t,1-n_t) + \lambda_t f_n(z,k_t,n_t) + \gamma_{2t} && (n_t)\\
0 &= -\theta_t + \beta\left[\lambda_{t+1}f_k(z,k_{t+1},n_{t+1}) + \theta_{t+1}(1-\delta_k)\right] + \gamma_{3t} && (k_{t+1})\\
0 &= -\lambda_t + \theta_t && (x_{k_t})\\
0 &= \lim_{T\to\infty}\beta^T\lambda_T k_{T+1} && (TVC)\\
f(z,k_t,n_t) &= c_t + x_{k_t} && \text{(Feasibility)}
\end{aligned}$$
Using complimentary slackness and interiority, and simplifying, these become
$$\begin{aligned}
u_c(c_t,1-n_t) &= \lambda_t\\
\lambda_t &= \lambda_{t+1}\beta\left[f_k(z,k_{t+1},n_{t+1}) + (1-\delta_k)\right]\\
f(z,k_t,n_t) &= c_t + x_{k_t}
\end{aligned}$$

If a steady state exists, it must satisfy these model equations – we must have that it satisfies the Euler equation:

$$u_c(c^\star, 1-n^\star) = u_c(c^\star, 1-n^\star)\beta\left[f_k(z,k^\star,n^\star) + (1-\delta_k)\right] \Longrightarrow 1 = \beta\left[f_k(z,k^\star,n^\star) + (1-\delta_k)\right]$$

It must also satisfy feasibility:

$$f(z,k^\star,n^\star) = c^\star + x^\star \underbrace{\Longrightarrow}_{\text{LoM of } k} f(z,k^\star,n^\star) = c^\star + \delta k^\star$$

For the Euler equation, it is necessary that $f_k(\cdot)$ be continuous, and also that the standard assumptions be true:

$$\lim_{k_t \to 0} f_k(\cdot) > \frac{1}{\beta} - 1 + \delta$$
$$\lim_{k_t \to \infty} f_k(\cdot) < \frac{1}{\beta} - 1 + \delta$$

If all of these conditions hold, a steady state will necessarily exist.

3. The skill premium is characterized as the ratio of the skilled wage w_s to the unskilled wage w_u, which are the marginal products of skilled and unskilled workers respectively:

$$w_u = \frac{\partial f}{\partial n_u} = z\left(\lambda(\mu(k_t)^\rho + (1-\mu)(n_s)^\rho)^{\frac{\sigma}{\rho}} + (1-\lambda)n_u^\sigma\right)^{\frac{1-\sigma}{\sigma}}(1-\lambda)n_u^{\sigma-1}$$

$$w_s = \frac{\partial f}{\partial n_s} = z\left(\lambda(\mu(k_t)^\rho + (1-\mu)(n_s)^\rho)^{\frac{\sigma}{\rho}} + (1-\lambda)n_u^\sigma\right)^{\frac{1-\sigma}{\sigma}}\lambda(\mu(k_t)^\rho + (1-\mu)(n_s)^\rho)^{\frac{\sigma-\rho}{\rho}}(1-\mu)n_s^{\rho-1}$$

so the skill premium is

$$\frac{w_s}{w_u} = \frac{\lambda}{1-\lambda}\frac{n_s^{\rho-1}}{n_u^{\sigma-1}}(1-\mu)(\mu(k_t)^\rho + (1-\mu)(n_s)^\rho)^{\frac{\sigma-\rho}{\rho}}$$

4. We have that countries differ in productivity z. If every country is in steady state, capital accumulation satisfies

$$k = \frac{x_k}{\delta_k} = \frac{f(z,k,n) - c}{\delta_k} \Longrightarrow \frac{k}{n_s} = \frac{f(z,k,n) - c}{n_s\delta_k}$$

so a country with higher z will have a higher capital / skill ratio, as f is increasing in z. Similarly, output per worker is defined by

$$\frac{y}{n} = \frac{f(z,k,n)}{n}$$

which is also increasing in z, so a country with higher productivity will have a higher output per worker. Finally, we have an equation for the skill premium above, and we can directly see that it is not affected by z, so it will be the same across countries with different productivities.

5. We now assume that countries differ in the ratio of skilled to unskilled workers $\frac{n_u}{n_s}$. As above, capital accumulation satisfies

$$k = \frac{x_k}{\delta_k} = \frac{f(z,k,n) - c}{\delta_k} \Longrightarrow \frac{k}{n_s} = \frac{f(z,k,n) - c}{n_s\delta_k}$$

so a country with a higher ratio of unskilled workers to skilled workers will have an increase in the marginal productivity to skilled labor (because the relative decrease in supply of skilled workers) which will lead to a decrease in the capital to skilled labor ratio, as they will demand more skilled labor. When the ratio of unskilled workers to skilled workers increases, output per worker will decrease because a higher proportion of unskilled workers will decrease $f(\cdot)$ in steady state. Finally, in our equation for the skill premium above, we can see that a greater proportion of unskilled workers will increase the skill premium, as the (relative) marginal product of skilled workers will increase.

6. As new technology appears, skill-biased technological change predicts that it will favor skilled workers over unskilled workers. In our model, we can see that the marginal product of skilled workers is increasing in z, as is the marginal product of unskilled workers. However, as long as the skill premium is greater than 1 (which, intuitively, it always should be), the marginal product of skilled workers increases *faster* in z than the marginal product of unskilled workers, so an increase in technology will lead to a greater increase in skilled wage than unskilled wage, an increase in the skill premium.

Transition paths in the one sector growth model

1. In the one-sector growth model, we have that the steady state level of capital is determined by the Euler equation
$$\beta[f'(k^\star) + (1-\delta)] = 1 \implies k^\star = \left(\frac{1/\beta - (1-\delta)}{\alpha \cdot A}\right)^{\frac{1}{\alpha-1}}$$
which is entirely in terms of our primitives. We also have from here k_0 directly. Steady state consumption is defined by the feasibility constraint
$$c^\star + \delta k^\star + \underbrace{g}_{=0} = f(k^\star) \implies c^\star = A(k^\star)^\alpha - \delta k^\star$$
We can guess an initial value for c_0, and iterate forward for N periods using the two difference equations, with k_0 given:
$$u'(c_{n+1})^{-1} = \beta u'(c_n)^{-1}[Af'(k_{n+1}) + (1-\delta)]$$
$$k_{n+1} = f(k_n) + (1-\delta)k_n - c_n$$
Our initial guess for c_0 is $f(k_0)$, and we use binary search to find the optimal c_0. Weirdly, the model will not converge over 600 periods. When we restrict to 150 periods it does, but over 600 capital always either goes to 0 or to the maximum. I am unsure why this is happening except that the shooting algorithm is finicky. I ended up with the optimal initial consumption of
$$c_0 = 1.08933574$$

2. I plotted the path from the initial condition to the steady state, also plotting the fixed points of the policy functions $C(K)$ and $K(C)$. The plot is Figure 1.

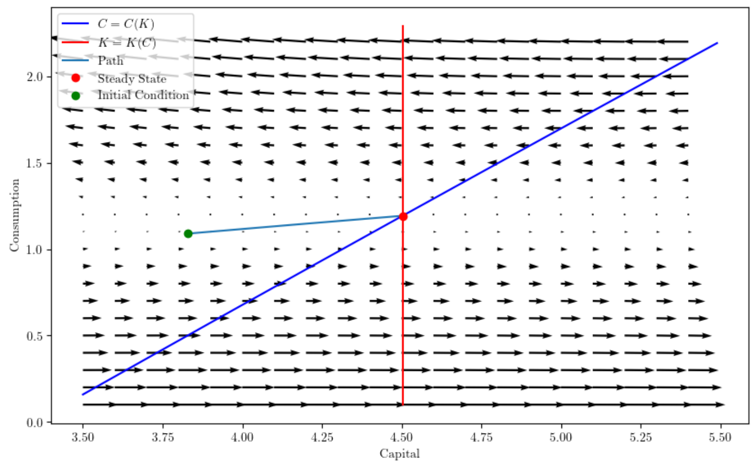

Figure 1: Phase Diagram

3. As agents get more impatient, they want to increase consumption in the current period rather than save capital for later. This leads, when they are significantly impatient, to no convergence. If there is convergence, it actually converges faster the more impatient agents are. A plot of the iterations to convergence on the value of β is Figure 2. I discretized β from 0.9 to 0.98. The empty values are where the model did not converge (including everything from 0.9 to 0.96). Convergence was slightly faster for some values just below the previous rate, but increased when it was far below.

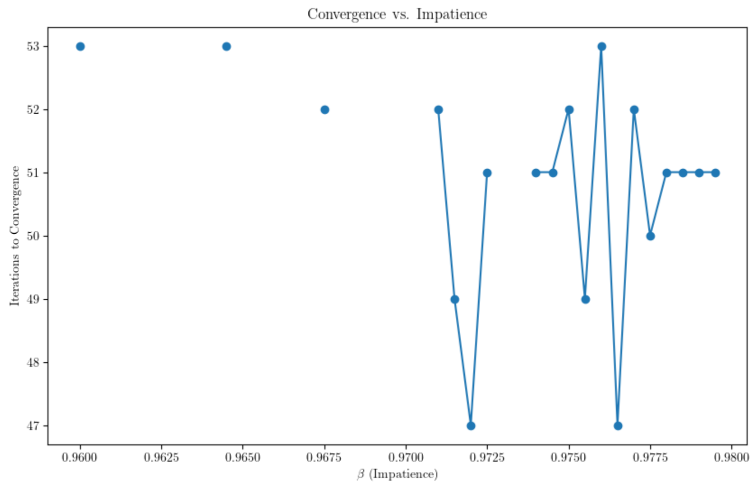

Figure 2: Convergence on Impatience

4. Similarly to class, we can write the one-sector growth model in recursive formulation as follows:

$$V(k) = \max_{(c,k')\in\Gamma(k)} \frac{c^{1-\sigma}}{1-\sigma} + \beta V(k')$$

where

$$\Gamma(k) = \{(c, k') : c \geq 0, k' \geq 0, c + k' \leq f(k) + (1-\delta)k\}$$

To perform the value function iteration, we guess an initial value function $V_0 = 0$, and discretize the capital and consumption spaces around the steady states. We then iteratively guess value functions until we converge to the optimal policy function $k'(k)$ and value function $V(k)$. The attained optimal capital accumulation function and the Euler residuals are in Figure 3.

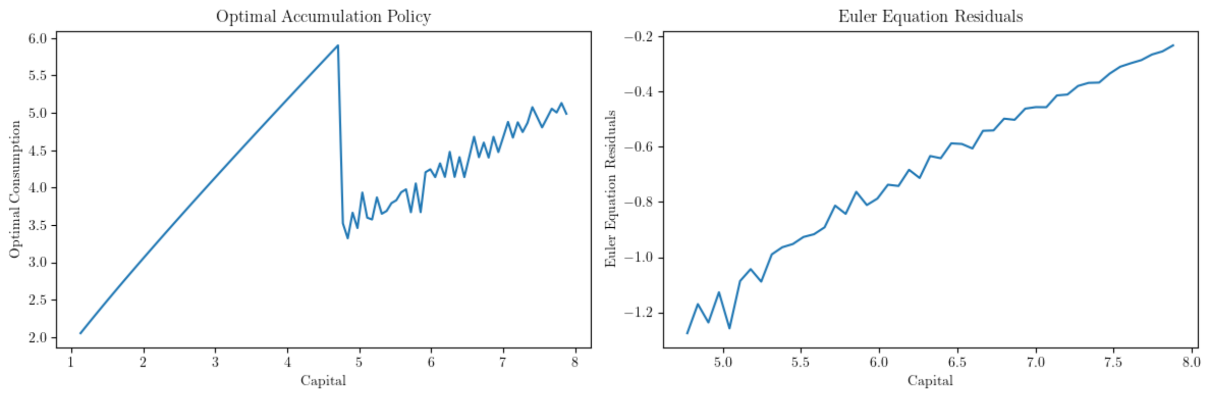

Figure 3: Optimal Accumulation Policy and Euler Equation Residuals

5. The policy function path and the shooting method path are in Figure 4, with the policy function path in blue, the shooting method path in orange, and the steady state in red. As we can see, the policy function path oscillates around the steady state significantly more than the shooting method path.

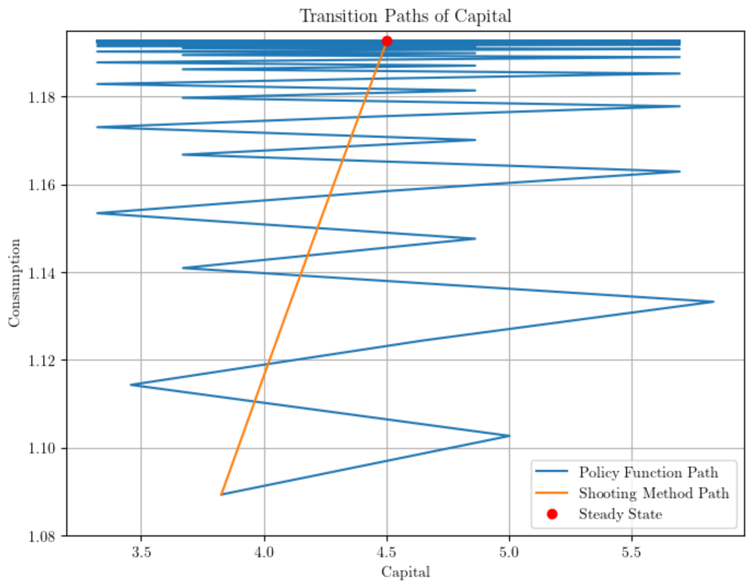

Figure 4: Transition Paths of Capital

My code for this problem is:

```python
# Preliminarites
import numpy as np
import matplotlib.pyplot as plt
plt.rc('text', usetex=True)
plt.rc('font', family='serif')


# Define model parameters
N = 600
alpha = 0.33
beta = 0.98
delta = 0.10
sigma = 0.95

# Define functions
def f(k):
    return np.where(k <= 0, 0, k ** alpha)

def f_prime(k):
    return np.where(k <= 0, 0, alpha * k ** (alpha - 1))

def f_prime_inv(x):
    return np.where(x <= 0, 0, (x / alpha) ** (1 / (alpha - 1)))
```

```
def u(c):
    return np.where(c <= 0, 0, c ** (1 - sigma) / (1 - sigma))

def u_prime(c):
    return np.where(c <= 0, 0, c ** (-sigma))

def u_prime_inv(x):
    return np.where(x <= 0, 0, x ** (-1 / sigma))

def next_k_c(k, c):
    k_next = f(k) + (1 - delta) * k - c
    return np.where(k_next <= 0, 0, k_next), np.where(k_next <= 0, 0,
       u_prime_inv(u_prime(c) / (beta * (f_prime(k_next) + (1 - delta)))))

def shooting(c0, k0, T=600):
    if c0 > f(k0) + (1 - delta) * k0:
        print("initial consumption is not feasible")
        return None

    c_vec = np.empty(T+1)
    k_vec = np.empty(T+2)

    c_vec[0] = c0
    k_vec[0] = k0

    for t in range(T):
        k_vec[t+1], c_vec[t+1] = next_k_c(k_vec[t], c_vec[t])

    k_vec[T+1] = f(k_vec[T]) + (1 - delta) * k_vec[T] - c_vec[T]

    return c_vec, k_vec

def bisection(k0, T=600, tol=1e-4, max_iter=500, k_ter=0, c_ter=0, verbose=
   False):
    # initial boundaries for guess c0
    c0_upper = 2*f(k0)
    c0_lower = 0
    c0 = (c0_upper + c0_lower) / 2

    i = 0
    while True:
        c_vec, k_vec = shooting(c0, k0, T)
        error = k_vec[-1] - k_ter
        if verbose:
            print("c:", c_vec[-1], "k:", k_vec[-1], "error:", error, "c0:", c0
                )

        # check if the terminal condition is satisfied
        if np.abs(error) < tol:
            if verbose:
                print('Converged successfully on iteration ', i+1)
            return c_vec, k_vec, c0, i+1

        i += 1
        if i == max_iter:
            if verbose:
                print('Convergence failed.')
            return c_vec, k_vec, c0, i+1

        # if iteration continues, updates boundaries and guess of c0
        if error > 0:
            c0_lower = c0
        else:
            c0_upper = c0


        c0 = (c0_lower + c0_upper) / 2
        if verbose:
            print(f"Updated bounds: c0_lower = {c0_lower}, c0_upper = {
               c0_upper}")



# Set steady states
k_star = f_prime_inv(1 / beta - (1 - delta))
c_star = f(k_star) - delta * k_star
k0 = 0.85 * k_star
c0 = 0.5 * f(k0)
print("k_star:", k_star, "k0:", k0, "c0:", c0, "f(k0):", f(k0), "c_star:",
   c_star)

# Test shooting with steady state
c_ss,k_ss = shooting(c_star, k_star, T=300)
print("C Difference:", c_ss[-1] - c_star)
print("K Difference:", k_ss[-1] - k_star)

c_vec, k_vec, c_init, _ = bisection(k0, T=150, k_ter=k_star, c_ter=c_star,
   verbose=False)



# Plot the phase diagram for capital and consumption
fig, ax = plt.subplots(figsize=(10, 6))


K_range = np.arange(3.5, 5.5, 0.01)
C_range = np.arange(1e-1, 2.3, 0.01)

# C tilde (fixed point equation)
ax.plot(K_range, [f(k) + (1 - delta) * k - f_prime_inv(1 / beta - 1 + delta)
   for k in K_range], color='b', label='$C=C(K)$')
```

```
# K tilde (fixed point equation)
ax.plot([f_prime_inv(1 / beta - 1 + delta) for c in C_range], C_range, color='
    r', label='$K=K(C)$')

# Stable branch
ax.plot(k_vec[:151], c_vec, label='Path')
ax.plot(k_star, c_star, 'ro', label='Steady State')
ax.plot(k0,c_init, 'go', label='Initial Condition')

K_grid = np.arange(3.5, 5.5, 0.1)
C_grid = np.arange(1e-1, 2.3, 0.1)


K_mesh, C_mesh = np.meshgrid(K_grid, C_grid)

next_K, next_C = next_k_c(K_mesh, C_mesh)
ax.quiver(K_grid, C_grid, next_K-K_mesh, next_C-C_mesh)

ax.set_xlabel('Capital')
ax.set_ylabel('Consumption')
ax.legend()
plt.savefig('macro_hw1_code/phase_diagram.png', bbox_inches='tight')



# See what happens if agents get more impatient
betas = np.arange(0.90, 0.98, 0.0005)
iters = np.empty(len(betas))

for i in range(len(betas)):
    beta = betas[i]
    _, _, _, iter = bisection(k0, T=150, k_ter=k_star, c_ter=c_star, verbose=
        False)
    if iter < 501:
        iters[i] = iter
    else:
        iters[i] = np.nan

# Plot impatience vs iterations
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(betas, iters, marker='o')

# Set labels and title
ax.set_xlabel(r'$\beta$ (Impatience)')
ax.set_ylabel('Iterations to Convergence')
ax.set_title('Convergence vs. Impatience')

plt.savefig('macro_hw1_code/iters.png', bbox_inches='tight')

# Fix beta
beta = 0.98
```

```
## Value function iteration

# Define functions
def value_function_iteration(k_grid, tol=1e-6, max_iter=1000):
    num_k = len(k_grid)
    v = np.zeros(num_k)

    for i in range(max_iter):
        v_new = np.zeros(num_k)
        for j in range(num_k):
            k = k_grid[j]
            c_grid = np.linspace(0, f(k) + (1 - delta) * k, num_k)
            v_next = np.zeros(num_k)
            for m in range(num_k):
                k_next = f(k) + (1 - delta) * k - c_grid[m]
                k_next_index = np.argmin(np.abs(k_grid - k_next))
                v_next[m] = u(c_grid[m]) + beta * v[k_next_index]
            v_new[j] = np.max(v_next)
        error = np.max(np.abs(v - v_new))
        v = v_new
        if error < tol:
            break
    return v

def optimal_policy(k_grid, v):
    num_k = len(k_grid)
    policy = np.zeros(num_k)
    for j in range(num_k):
        k = k_grid[j]
        c_grid = np.linspace(0, f(k) + (1 - delta) * k, num_k)
        v_next = np.zeros(num_k)
        for m in range(num_k):
            k_next = f(k) + (1 - delta) * k - c_grid[m]
            k_next_index = np.argmin(np.abs(k_grid - k_next))
            v_next[m] = u(c_grid[m]) + beta * v[k_next_index]
        policy[j] = c_grid[np.argmax(v_next)]

    return policy

def euler_residuals(policy, k_grid):
    num_k = len(k_grid)
    residuals = np.zeros(num_k)
    for j in range(num_k):
        k = k_grid[j]
        c = policy[j]
        k_next = f(k) + (1 - delta) * k - c
        c_next = np.interp(k_next, k_grid, policy)
        residuals[j] = u(c) - beta * u(c_next) * (alpha * k_next**(alpha-1) +
           1 - delta)

    return residuals
```

```
# Grid for capital
k_grid = np.linspace(0.25*k_star, 1.75*k_star, 101)


# Iterate
v = value_function_iteration(k_grid, tol=1e-6, max_iter=1000)
policy = optimal_policy(k_grid, v)
residuals = euler_residuals(policy, k_grid)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

ax1.plot(k_grid, policy)
ax1.set_xlabel('Capital')
ax1.set_ylabel('Optimal Consumption')
ax1.set_title('Optimal Accumulation Policy')

ax2.plot(k_grid, residuals)
ax2.set_xlabel('Capital')
ax2.set_ylabel('Euler Equation Residuals')
ax2.set_title('Euler Equation Residuals')

plt.tight_layout()
plt.savefig('macro_hw1_code/value_function_iteration.png', bbox_inches='tight'
    )


# Generate transition path from policy function
def generate_transition_path(k0, policy, k_grid, T):
    path = np.zeros(T)
    path[0] = k0
    for t in range(1, T):
        k_index = np.argmin(np.abs(k_grid - path[t-1]))
        path[t] = policy[k_index]
    return path

policy_path = generate_transition_path(k0, policy, k_grid, 151)



# Plot the transition paths
plt.figure(figsize=(8, 6))
plt.plot(policy_path, c_vec, label='Policy Function Path')
plt.plot(k_vec[:151], c_vec, label='Shooting Method Path')
plt.plot(k_star, c_star, 'ro', label='Steady State')
plt.xlabel('Capital')
plt.ylabel('Consumption')
plt.ylim(1.08, 1.195)
plt.title('Transition Paths of Capital')
plt.legend()
plt.grid(True)
plt.savefig('macro_hw1_code/transition_paths.png', bbox_inches='tight')
```