STOCHASTIC OPTIMAL GROWTH MODEL

MODEL

Consider the following social planner's problem:

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

s.t $c_t + k_{t+1} = z_t f(k_t)$, where productivity z_t is a stochastic process. $k_0 > 0$ and $z_0 > 0$ are given.

TIMELINE

1. At the beginning of time t, the exogenous shock z_t is realized.
2. Thus we know the pair (k_t, z_t) and current output $z_t k_t^\alpha$. (k_t, z_t) is called the state of the economy.
3. When consumption c_t is chosen then at the end of period capital k_{t+1} is accumulated.

HISTORY OF THE SHOCK

- Let z^t denote a history of realizations of the shock

$$z^t := (z_0, z_1, ..., z_t) = (z^{t-1}, z_t) .$$

- Let $c_t(z^t)$ and $k_{t+1}(z^t)$ denote contingent plans for consumption and capital accumulation conditional on z^t.

EXPECTATION

- z^t is unknown at $t = 0$, hence $c_t(z^t)$ is unknown at $t = 0$.
- the (time-0 view) objective function is uncertain at $t = 0$, hence expectation is used:

$$\mathbb{E} \sum_{t=0}^{\infty} \beta^t u(c_t) = \sum_{t=0}^{\infty} \sum_{z^t} \beta^t \pi(z^t) u(c_t(z^t)),$$

if assuming discrete states of z_t, where $\pi(z^t)$ denotes the probability of z^t.

SEQUENCE PROBLEM RESTATED

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

s.t.

$$c_t(z^t) + k_{t+1}(z^t) = z_t(z^t) f(k_t(z^{t-1})), \qquad \forall\ z^t,$$

and given $\pi(z^t) \geq 0$, $k_0 > 0$, $z_0 > 0$.

LAGRANGIAN APPROACH

- Denote $\lambda_t(z^t) \geq 0$ as the stochastic multiplier for each constraint

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

- FOC

$$\begin{aligned} c_t(z^t)&: \quad \beta^t \pi(z^t) u'(c_t(z^t)) = \lambda_t(z^t) \\ k_{t+1}(z^t)&: \quad \lambda_t(z^t) = \sum_{z_{t+1} \mid z^t} \lambda_{t+1}(z^t, z_{t+1})[z_{t+1} f'(k_{t+1}(z^t))] \end{aligned}$$

where the realization of z_{t+1} may depend on the history z^t (serial correlation).

EULER EQUATION

- Eliminating the multipliers gives

$$
\begin{aligned}
u'(c_t(z^t)) &= \beta \sum_{z_{t+1} \mid z^t} u'(c_{t+1}(z^t, z_{t+1}))[z_{t+1} f'(k_{t+1}(z^t))] \frac{\pi(z^t, z_{t+1})}{\pi(z^t)} \\
&= \beta \sum_{z_{t+1} \mid z^t} u'(c_{t+1}(z^t, z_{t+1}))[z_{t+1} f'(k_{t+1}(z^t))] \pi(z_{t+1} \mid z^t)
\end{aligned}
$$

- This is the stochastic version of the Euler equation

$$
u'(c_t) = \beta\, \mathbb{E}[u'(c_{t+1}) z_{t+1} f'(k_{t+1}) \mid z^t]
$$

RECURSIVE FORMULATION

- The Bellman equation can be written as

$$
\begin{aligned}
v(k_t, z_t) &= \max_{c_t, k_{t+1}} \left[u(c_t) + \beta\, \mathbb{E}[v(k_{t+1}, z_{t+1}) \mid z^t] \right] \\
&= \max_{c_t, k_{t+1}} \left[u(c_t) + \beta \sum_{z_{t+1} \mid z^t} v(k_{t+1}, z_{t+1}) \pi(z_{t+1} \mid z^t) \right]
\end{aligned}
$$

s.t $c_t + k_{t+1} = z_t f(k_t)$.

- FOC and envelope condition

$$
\begin{aligned}
u'(c_t) &= \beta\, \mathbb{E}[v_k(k_{t+1}, z_{t+1}) \mid z^t] \\
v_k(k_t, z_t) &= u'(c_t) z_t f'(k_t)
\end{aligned}
$$

- Eliminating v_k we have

$$
u'(c_t) = \beta\, \mathbb{E}[u'(c_{t+1}) z_{t+1} f'(k_{t+1}) \mid z^t]
$$

TWO SPECIAL CASES

- i.i.d.: $\pi(z_{t+1} \mid z^t) = \pi(z)$
- Markov process: $\pi(z_{t+1} \mid z^t) = \pi(z_{t+1} \mid z_t)$

CASE 1: I.I.D.

- $u(c) = \ln c$
- $f(k) = k^{\alpha}$
- Suppose the productivity z_t is an i.i.d. sequence which takes values uniformly distributed in a set of discrete points:

$$z_t \in \{0.9792, 0.9896, 1.0000, 1.0106, 1.0212\}.$$

SOLUTION

- Analytical solution

$$
\begin{aligned}
k_{t+1} &= \alpha\beta z_t k_t^{\alpha}, \\
c_t &= (1-\alpha\beta) z_t k_t^{\alpha},
\end{aligned}
$$

which will allow us to assess the accuracy of the solution we compute.

- Note that $\{k_t\}$ will **not** converge to a fixed point because $\{z_t\}$ is random. But we can still calculate its expection:

$$
k^* := \lim_{t\to\infty} \mathbb{E}_0(k_t) = \frac{\ln(\alpha\beta) + \mathbb{E}(\ln z)}{1-\alpha}.
$$

CALIBRATION

- $\alpha = 1\,/\,3$
- $\beta = 0.95$
- $z_t \in \{0.9792, 0.9896, 1.0000, 1.0106, 1.0212\}$.
- $k^* = (\alpha\beta)^{\frac{1}{1-\alpha}}$
- $k \in \{0.95k^*, 1.05k^*\}$

CODE FRAGMENTS

- import

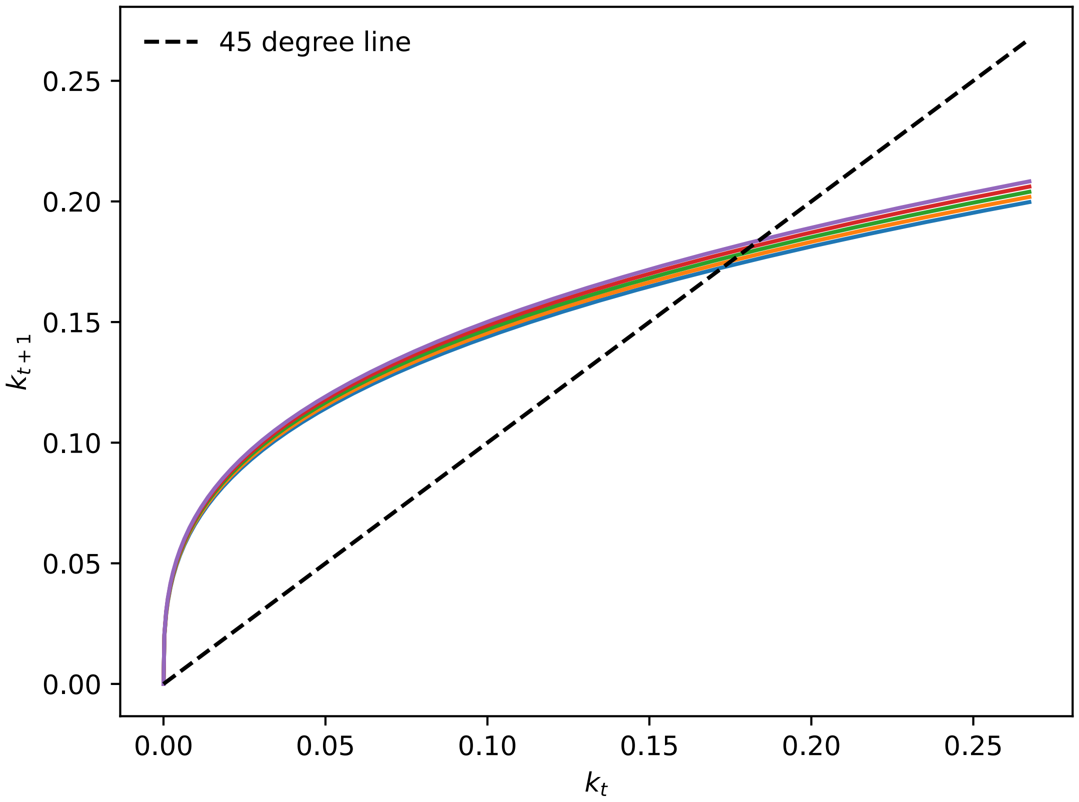
```
import timeit
import numpy as np
import matplotlib.pyplot as plt
```

- parameter

```
beta = 0.95
alpha = 1 / 3
kss = (alpha * beta) ** (1 / (1 - alpha))
kmin = 0.95 * kss
kmax = 1.05 * kss
n_k = 100
kgrid = np.linspace(kmin, kmax, n_k)
zgrid = np.array([0.9792, 0.9896, 1.0000, 1.0106, 1.0212])
P = np.array([0.2, 0.2, 0.2, 0.2, 0.2])  # probability of z
n_z = len(zgrid)
```

ANALYTICAL POLICY FUNCTION

```
g_analytical = np.outer(alpha*beta*kgrid**alpha, zgrid)  # vectorized calculation
```



ORGANIZE THE FUNCTION (1)

- How can we organize the value function, for example, $V(k, z)$ (as well as the policy function) if there are two or more state variables?
    - treat V as a matrix
    - treat V as a vector

ORGANIZE THE FUNCTION (2)

- as a matrix: For example, use k grids as row index, and z grids as column index.
  - What if there are many random state variables? Suppose at each period we have two (or more) random state variables: $s = (s_1, s_2)$ and $z = (z_1, z_2, z_3)$. We can define a new (composite) state variable $x := s \otimes z$ (Kronecker product):

$$\begin{aligned} x &= (x_1, x_2, x_3, x_4, x_5, x_6) \\ &= ((s_1, z_1), (s_1, z_2), (s_1, z_3), (s_2, z_1), (s_2, z_2), (s_2, z_3)) \end{aligned}$$

ORGANIZE THE FUNCTION (3)

- as a vector: we can use Kronecker product for all state variables (both k and z):
  - define a new (composite) state variable $x := k \otimes z$, calculate the new function $V_{vector}(x)$
  - In this manner, the function with multiple state variables can be uniformly expressed.
  - Restore $V(k, z)$ from $V_{vector}(x)$: `i_x = i_k * n_z + i_z`, where `i` denotes the index (location), and `n` denotes the total number. Either use a loop

```
for i_x in range(n_k * n_z):
    i_k = i_x // n_z   # quotient
    i_z = i_x % n_z   # remainder
    V[i_k,i_z] = V_vector(i_x)
```

Or,

```
V=V_vector.reshape(n_k,n_z)   # by row
```

CODE FRAGMENTS

- The following are code examples treating the value function `V_next` as a matrix (row is k and column is z):

```
def V_current(k_next, k_index, z_index, V_next):
    """objective value function to be maximized"""
    c = zgrid[z_index]*kgrid[k_index]**alpha - k_next
    EV = np.sum(P*V_next[k_index,:])  # expectation
    res = u(c) + beta * EV
    return res

def V_max(k_index, z_index, V_next)
    """loop over possible k_next to max v"""
    ...
```

CODE FRAGMENTS

```
def V_update(V):
    V_new = np.zeros_like(V)
    g_new = np.zeros_like(V)
    for i_z in range(n_z):  # loop over all state z
        for i_k in range(n_k):  # loop over all state k
            V_new[i_k, i_z], g_new[i_k, i_z] = V_max(i_k, i_z, V)
    return V_new, g_new
```

- If using the concavity and monotonicity tricks, then the z loop must be executed first!
- If using the monotonicity trick, then the z loop can still be parallelized!

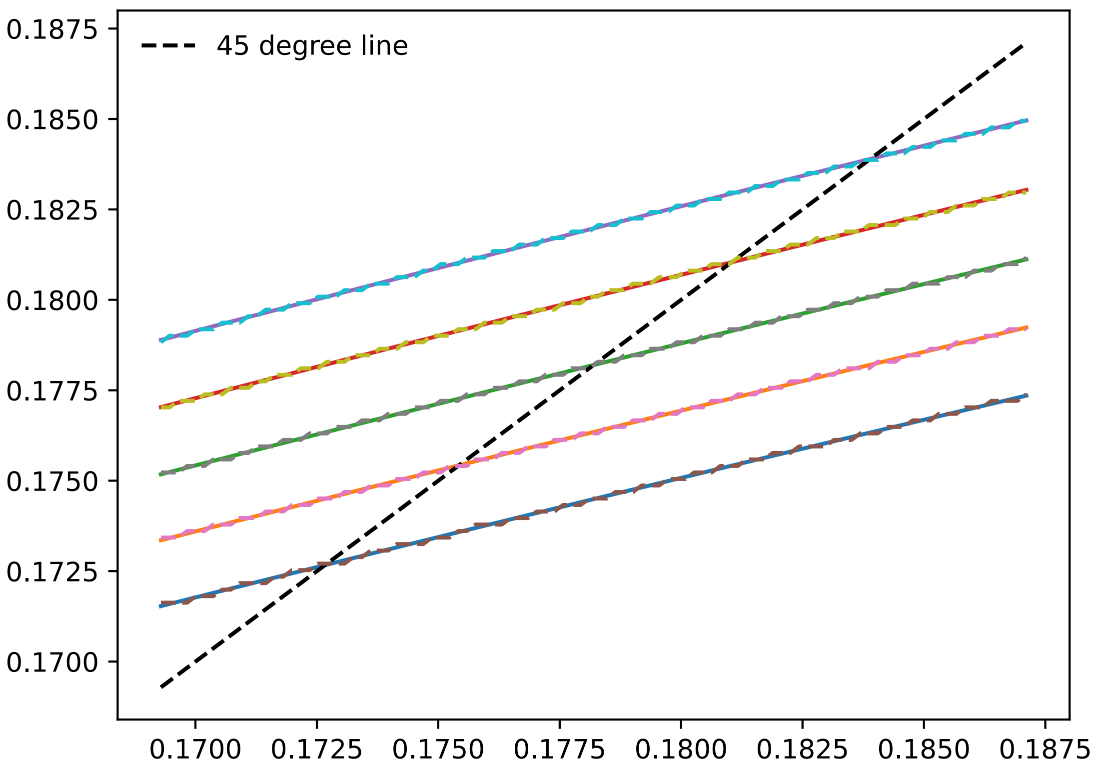
TEST

```
V0 = np.zeros((n_k, n_z))
start_time = timeit.default_timer()
V, g = V_iteration(V0)
print("The time difference is :", timeit.default_timer() - start_time)
```

```
Error at iteration 50 is: 0.3875261474470548
Error at iteration 100 is: 0.02981818983440121
Error at iteration 150 is: 0.0022943598796203446
Error at iteration 200 is: 0.000176539642241347
Error at iteration 250 is: 1.358382472460562e-05
Error at iteration 300 is: 1.0452070711153283e-06

Converged in 301 iterations.
The time difference is : 0.0314454699982889
```

PLOT



45 degree line
0.1875
0.1850
0.1825
0.1800
0.1775
0.1750
0.1725
0.1700
0.1700
0.1725
0.1750
0.1775
0.1800
0.1825
0.1850
0.1875

CASE 2: MARKOV PROCESS

- A (first-order) Markov process has the property that, conditional on the current z_t , future realizations are independent of z^{t-1} . In this sense, the current z_t is a sufficient statistic for the past.
- Markov processes are recursive, and so are a natural setting for dynamic programming approaches.
- Consider z_t with discrete support, usually referred to as a Markov chain. Markov processes with continuous support have a similar structure.

MARKOV CHAIN (1)

- A finite Markov chain $\{z_t\}$ is a triple (z, P, ψ_0).
  - z is an n-vector listing the possible states (outcomes) of the chain
    - A realization of z_t takes on the value of one of the states in z.
  - P is an $n \times n$ probability transition matrix $[p_{ij}]_{n \times n}$.
  - $p_{ij} = \text{Prob}[z_{t+1} = z_j \mid z_t = z_i]$, $0 \le p_{ij} \le 1$, $\sum_j p_{ij} = 1$ for all i.
  - ψ_0 is an n-vector initial distribution over the states.
    - $\psi_{0,i} = \text{Prob}[z_0 = z_i]$, $0 \le \psi_{0,i} \le 1$, $\sum_i \psi_{0,i} = 1$.

MARKOV CHAIN (2)

- Suppose the productivity z_t takes values in a 5-point Markov chain

$$z_t \in \{0.9792, 0.9896, 1, 0000, 1.0106, 1.0212\}$$

with transition matrix :

$$\Pi = \begin{pmatrix} 0.9727 & 0.0273 & & & \\ 0.0041 & 0.9806 & 0.0153 & & \\ & 0.0082 & 0.9837 & 0.0081 & \\ & & 0.0153 & 0.9806 & 0.0041 \\ & & & 0.0273 & 0.9727 \end{pmatrix}.$$

CODE FRAGMENTS

```
def V_current(k_next, k, z_index, V_next):
    c = budget(k_next, k, z_index)
    EV = np.sum(P[z_index]*V_next)
    res = u(c) + beta * EV
    return res
```

NEXT QUESTION

- Now we have solved the policy function $k_{t+1} = g(k_t, z_t)$ numerically.
- We cannot expect $\{k_t\}$ to converge to some steady state k^*. What about the limiting distribution of k_t when $t \to \infty$?
- A stochastic analogue to a steady state of a deterministic system is a stationary (invariant) distribution.
    - Does the sequence of k_t converge to a stationary limiting distribution? If so, what does the limiting distribution look like?