

Lecture 7: Overlapping-generations models (OLG)

FIE463: Numerical Methods in Macroeconomics and Finance using Python

Richard Foltyn
NHH Norwegian School of Economics

March 3, 2026

See GitHub repository for notebooks and data:

<https://github.com/richardfoltyn/FIE463-V26>

Contents

1 Overlapping-generations model	1
1.1 Steady-state equilibrium	1
1.2 Transition dynamics	9

1 Overlapping-generations model

In previous lectures, we studied a simple two-period consumption-savings problem in *partial equilibrium*. This can be taken to a general-equilibrium setting if we assume that at each point in time, there are two generations alive: one young and one old.

Additionally, we extend the model with the same production sector that we introduced in the previous lecture, i.e., perfectly competitive firms with Cobb-Douglas technology which use capital K and labor L . Unlike in the previous lecture, we now study a setting with *endogenous* capital which arises from the households' savings decision, but we fix labor supply to be exogenous.

Additional material

- [The Overlapping Generations Model](#) on QuantEcon provides an alternative exposition of this topic.

1.1 Steady-state equilibrium

We first study the steady-state equilibrium, i.e., an equilibrium where all quantities and prices are constant over time. If the economy starts in a steady state, it will remain there as long as the parameters don't change. In the last section, we extend this analysis and study transition dynamics when the economy experiences an unanticipated shock to TFP which takes it away from the steady state.

1.1.1 Household problem

At each point in time, the economy is populated by N identical young and N identical old households. Without loss of generality, we set $N = 1$. Each cohort solves a two-period consumption-savings problem which we encountered earlier,

$$\begin{aligned} \max_{c_y, c_o, a} \quad & \left\{ u(c_y) + \beta u(c_o) \right\} \\ \text{s.t.} \quad & c_y + a = w \\ & c_o = (1 + r)a \\ & c_y \geq 0, c_o \geq 0, a \geq 0 \end{aligned}$$

where β is the discount factor, r is the interest rate, w is the wage income received when young, a are savings by the young, and (c_y, c_o) is the optimal consumption allocation when young and old, respectively. Per-period utility $u(c)$ is the CRRA utility function given by

$$u(c) = \begin{cases} \frac{c^{1-\gamma}}{1-\gamma} & \text{if } \gamma \neq 1 \\ \log(c) & \text{if } \gamma = 1 \end{cases}$$

where γ is the RRA coefficient and $\log(\bullet)$ denotes the natural logarithm.

We assume that the household inelastically supplies one unit of labor when young, therefore w is both the wage rate and their labor income. We impose that the household does not work when old and therefore needs to rely on its savings to finance consumption.

1.1.2 Firm problem

The firm problem is almost the same as the one we studied in the previous lecture: Firms combine capital K and labor L in a [Cobb-Douglas production function](#) to produce output Y ,

$$Y = zK^\alpha L^{1-\alpha}$$

and maximize profits Π according to

$$\max_{K, L} \Pi = zK^\alpha L^{1-\alpha} - (r + \delta)K - wL$$

where $(r + \delta)K$ is the cost of capital and wL is the cost of labor. We assume that capital depreciates by a fraction δ each period so that r is the net return on capital after depreciation.

The solution is characterized by the firm's first-order conditions

$$\begin{aligned} r + \delta &= \alpha z \left(\frac{K}{L} \right)^{\alpha-1} = \alpha z k^{\alpha-1} \\ w &= (1 - \alpha) z \left(\frac{K}{L} \right)^\alpha = (1 - \alpha) z k^\alpha \end{aligned} \tag{1}$$

where we define the capital-labor ratio as $k \equiv \frac{K}{L}$. We'll use these conditions in the numerical solution below.

1.1.3 Equilibrium

The general equilibrium in this economy is a set of quantities (K, L, Y, c_o, c_y, a) and prices (r, w) which solve the household and firm problems such that the following market clearing conditions are satisfied:

- Asset market: $K = Na$ (capital K demanded by firms equals aggregate savings Na supplied by households).
- Labor market: $L = N$ (labor L demanded by firms equals exogenously supplied labor by households).

- Goods market: $Y + (1 - \delta)K = N(c_y + c_o + a)$ (aggregate output & undepreciated capital equals the amount of goods consumed by young and old each period and investment by the young).

Due to Walras' Law, we only have to make sure that two of these markets clear as this implies market clearing in the residual market.

Note: The goods market clearing condition might differ from the one you are used to, $Y_t = C_t + I_t$, where aggregate output is equal to aggregate consumption C_t and investment I_t . However, these two conditions are equivalent since aggregate consumption is $C = N(c_y + c_o)$, and the next-period capital stock is $K_{t+1} = Na$. Plugging these expressions into the goods market clearing condition from above, we have

$$Y_t + (1 - \delta)K_t = C_t + K_{t+1} \implies Y_t = C_t + \underbrace{K_{t+1} - (1 - \delta)K_t}_{I_t}$$

1.1.4 Some useful analytical results

To solve the problem numerically, we explore an alternative to what we did in the previous lectures where we ran a minimizer to solve the household problem. Instead, we use the household's first-order conditions to find an analytical expression for the savings rate s so we don't have to run a minimizer.

Let the *savings rate* s be the fraction of resources saved when young, i.e.,

$$\begin{aligned} a &= sw \\ c_y &= (1 - s)w \\ c_o &= (1 + r)sw \end{aligned}$$

We can then rewrite the maximization problem in terms of the single variable s ,

$$\max_{s \in [0,1]} u(\underbrace{(1 - s)w}_{c_y}) + \beta u(\underbrace{(1 + r)sw}_{c_o})$$

We can ignore the constraints $s \geq 0$ and $s \leq 1$ since we know that consuming zero either when young or old will result in $-\infty$ utility, so these constraints won't be binding. The Lagrangian for this problem is therefore just the lifetime utility,

$$\mathcal{L} = u((1 - s)w) + \beta u((1 + r)sw)$$

Taking the derivative with respect to s , we see that

$$\frac{\partial \mathcal{L}}{\partial s} = -u'((1 - s)w)w + \beta u'((1 + r)sw)(1 + r)w = 0$$

This is the usual *Euler equation* that we would expect to find for this type of problem, but expressed in terms of the savings rate s :

$$u'(\underbrace{(1 - s)w}_{c_y}) = \beta(1 + r)u'(\underbrace{(1 + r)sw}_{c_o})$$

Using the functional form for the utility function, the Euler equation becomes

$$((1 - s)w)^{-\gamma} = \beta(1 + r)((1 + r)sw)^{-\gamma}$$

After some manipulations, this can be solved for optimal s :

$$s = \frac{1}{1 + \beta^{-\frac{1}{\gamma}}(1 + r)^{1 - \frac{1}{\gamma}}} \quad (2)$$

We'll use this expression to compute capital supply by households for a given interest rate.

1.1.5 Numerical solution

Solution algorithm

The structure of the solution algorithm is similar to the previous lecture. This time, we opt to find the equilibrium capital-labor ratio $k = \frac{K}{L}$, but we could have just as well opted to iterate over one of the equilibrium prices r or w .

Our implementation to find the general equilibrium proceeds as follows:

1. Define the problem's parameters.
2. Write a function that computes prices (r, w) for a given k (use the firm's first-order conditions from (1)).
3. Write a function that solves the household problem for given r and returns the optimal savings rate (use the analytical solution (2)).
4. Write a function $f(k)$ that returns the excess demand for capital $K - Na$ for a given capital-labor ratio k .

Use the functions defined in steps (2) and (3) for this purpose.

5. Call a root finder to locate the root of f where $f(k^*) = 0$. The root finder will repeatedly call f to locate the equilibrium k^* .
6. Once the root finder terminates and returns the equilibrium k^* , compute and store all other equilibrium quantities and prices.

Implementation

```
[1]: # Enable automatic reloading of external modules
%load_ext autoreload
%autoreload 2
```

Step 1: Problem parameters

The full implementation is provided in the file `lecture07_olg.py`. For expositional convenience, the following code segments replicate selected code blocks from that file.

We define a `dataclass` called `Parameters` to store the problem parameters:

```
[2]: from dataclasses import dataclass

@dataclass
class Parameters:
    """
    Parameters for the overlapping generations model.
    """
    alpha: float = 0.36      # Capital share in production function
    delta: float = 1.0       # Depreciation rate (full depreciation)
    z: float = 1.0           # TFP
    beta: float = 0.96**30   # Discount factor
    gamma: float = 2.0       # RRA in utility
    N: int = 1               # Number of households per cohort
```

Since there are only two adult generations alive at any point in time, it seems reasonable to assume that one period corresponds to roughly 30 years. This is reflected in the choice of parameters:

- If we apply the standard discount factor at *annual* frequency of 0.96 over 30 years, the resulting discount factor for this calibration should be $\beta = 0.96^{30}$, i.e., the annual discount factor taken to the power of 30.

- Similarly, if capital depreciates by a few percent a year, over 30 years the capital stock will have almost fully depreciated, so for simplicity we set $\delta = 1$ to reflect that (you can verify that an annual depreciation of 6% results in a depreciation of $1 - (1 - 0.06)^{30} \approx 0.84$ over 30 years).

We can now define an instance of `Parameters` to be used below.

```
[3]: # Create parameter instance
par = Parameters()
```

Step 2: Compute equilibrium prices from k

The root finder will ask us to evaluate excess capital demand for each conjectured k . First, we implement the following function to map k to factor prices r and w using the firm's first-order conditions.

```
[4]: def compute_prices(k, par: Parameters):
    """
    Return factor prices for a given capital-labor ratio and parameters.

    Parameters
    -----
    k : float
        Capital-labor ratio
    par : Parameters
        Parameters for the given problem

    Returns
    -----
    r : float
        Return on capital after depreciation (interest rate)
    w : float
        Wage rate

    """

    # Return on capital after depreciation (interest rate)
    r = par.alpha * par.z * k**(par.alpha - 1) - par.delta

    # Wage rate
    w = (1-par.alpha) * par.z * k**par.alpha

    return r, w
```

Step 3: Solve the household problem

The second building block required for the root finder is the solution to the household problem. We define the following function which returns the household's optimal savings rate given r :

```
[5]: def compute_savings_rate(r, par: Parameters):
    """
    Compute the savings rate using the analytical solution
    to the household problem.

    Parameters
    -----
    r : float
        Return on capital after depreciation (interest rate)
    par : Parameters
        Parameters for the given problem

    Returns
```

```

-----
s : float
    Savings rate
"""

s = 1/(1 + par.beta**(-1/par.gamma) * (1+r)**(1-1/par.gamma))

return s

```

Your turn.

Plot the household's optimal savings rate s returned by `compute_savings_rate()` for r on the interval $[0.01, 0.2]$. Does the shape intuitively make sense (given the parameter for the risk aversion γ)?

Step 4: Compute excess capital demand

We can now combine the return values from these functions to compute excess demand for capital, implemented in the function below. This function will be called by the root finder to find the equilibrium k^* .

```

[6]: def compute_capital_ex_demand(k, par: Parameters):
    """
    Compute the excess demand for capital.

    Parameters
    -----
    k : float
        Capital-labor ratio
    par : Parameters
        Parameters for the given problem

    Returns
    -----
    ex_demand : float
        Excess demand for capital
    """

    # Compute prices from firm's FOCs
    r, w = compute_prices(k, par)

    # Compute savings rate
    srate = compute_savings_rate(r, par)

    # Aggregate supply of capital by households (savings)
    A = srate * w * par.N

    # Aggregate labor supply
    L = par.N

    # Aggregate capital demand
    K = k * L

    # Excess demand for capital
    ex_demand = K - A

    return ex_demand

```

Your turn.

Plot the function `compute_capital_ex_demand()` for k on the interval $[0.01, 0.3]$ to verify that the function indeed has a root.

Step 5: Call the root finder

We can now test the code by calling the root finder. We use the default 'brentq' method, but could just as well have opted for a Newton-based algorithm.

```
[7]: from scipy.optimize import root_scalar

# Initial bracket for k used by root finder
bracket = (1.0e-3, 1)

# Call root finder. Pass Parameters using args argument.
res = root_scalar(
    compute_capital_ex_demand, bracket=bracket, args=(par, )
)
```

Inspecting the result returned by the root finder shows that the algorithm terminated successfully:

```
[8]: res

[8]:      converged: True
      flag: converged
      function_calls: 14
      iterations: 13
      root: 0.06615197337785264
      method: brentq
```

Step 6: Compute remaining equilibrium quantities

It is convenient to wrap the root finder into an additional function which also computes the equilibrium values and returns these as an instance of `SteadyState`. These are stored in the dedicated data class `SteadyState`, defined below:

```
[9]: @dataclass
class SteadyState:
    """
    Steady-state equilibrium of the OLG model.
    """
    par: Parameters = None      # Parameters used to compute equilibrium
    c_y: float = None          # Consumption when young
    c_o: float = None          # Consumption when old
    a: float = None            # Savings when young
    s: float = None            # Savings rate when young
    r: float = None            # Interest rate (return on capital)
    w: float = None            # Wage rate
    K: float = None            # Aggregate capital stock
    L: float = None            # Aggregate labor demand
    I: float = None            # Aggregate investment
    Y: float = None            # Aggregate output
```

The following function calls the root finder, computes the equilibrium values, and returns these as an instance of `SteadyState`:

```
[10]: def compute_steady_state(par: Parameters):
    """
    Compute the steady-state equilibrium for the OLG model.
```

```

Parameters
-----
par : Parameters
    Parameters for the given problem

Returns
-----
eq : SteadyState
    Steady state equilibrium of the OLG model
"""

# Find the equilibrium k=K/L with a root finder. Excess demand for capital
# has to be zero in equilibrium.
res = root_scalar(
    compute_capital_ex_demand, bracket=(1.0e-3, 1), args=(par, )
)

if not res.converged:
    print('Equilibrium root finder did not terminate successfully')

# Equilibrium K
K = res.root * par.N

# Create instance of equilibrium class
eq = SteadyState(par=par, K=K, L=par.N)

# Equilibrium prices
eq.r, eq.w = compute_prices(eq.K / eq.L, par)

# Investment in steady state
eq.I = eq.K * par.delta

# Equilibrium household choices
eq.s = compute_savings_rate(eq.r, par)
eq.a = eq.s * eq.w
eq.c_y = eq.w - eq.a
eq.c_o = (1 + eq.r) * eq.a

# Equilibrium output
eq.Y = par.z * eq.K**par.alpha * eq.L**(1-par.alpha)

# Aggregate consumption
C = par.N * (eq.c_y + eq.c_o)
# Check that goods market clearing holds using Y = C + I
assert abs(eq.Y - C - eq.I) < 1.0e-8

return eq

```

```

[11]: # Compute equilibrium, store as equilibrium instance
eq = compute_steady_state(par)

```

The module `lecture07_olg.py` implements a helper function `print_steady_state()` which can be used to report the equilibrium quantities in a nicely formatted fashion:

```

[12]: from lecture07_olg import print_steady_state

# Print equilibrium allocation & prices
print_steady_state(eq)

```

```

Steady-state equilibrium:
Households:
  c_y = 0.17460

```



```

c_o = 0.13542
a = 0.06615
Firms:
K = 0.06615
L = 1.00000
Y = 0.37618
Prices:
r = 1.04718
w = 0.24076
Market clearing:
Capital market: -6.93889e-17
Goods market: 1.66533e-16

```

Your turn.

You are interested in how the equilibrium prices depend on the cohort size N . Plot the equilibrium prices r and w when varying N over the range of integers from 1 to 10.

1.2 Transition dynamics

In the previous section we solved for the steady state, where the economy would remain in the absence of any exogenous changes. We now investigate what happens when the economy is hit by unanticipated shocks (so-called “MIT shocks”). Because these shocks are unanticipated, households don’t form expectations over them, and once a shock is realized, households have perfect foresight of the dynamic path the economy takes towards a new steady state (full information rational expectations, a common assumption in macroeconomics).

1.2.1 Household problem

The household problem is unchanged from before, but now all quantities have additional time indices t as prices and optimal choices are allowed to change along the transition path:

$$\begin{aligned}
& \max_{c_{y,t}, c_{o,t+1}, a_t} \left\{ u(c_{y,t}) + \beta u(c_{o,t+1}) \right\} \\
& \text{s.t. } c_{y,t} + a_t = w_t \\
& \quad c_{o,t+1} = (1 + r_{t+1})a_t \\
& \quad c_{y,t} \geq 0, c_{o,t+1} \geq 0, a_t \geq 0 \text{ for all } t
\end{aligned}$$

Note that consumption when old now explicitly depends on the interest rate *next period*, r_{t+1} , which can be different from the interest rate in t .

1.2.2 Firm problem

As firms solve a purely static problem, it remains unchanged from earlier.

1.2.3 Equilibrium

The equilibrium definition is also unchanged from before, but we explicitly add time indices t . The general equilibrium in this economy is a set of quantities $(K_t, L_t, Y_t, c_{o,t}, c_{y,t}, a_t)$ and prices (r_t, w_t) which solve the household and firm problems such that the following market clearing conditions are satisfied:

- Asset market: $K_t = Na_{t-1}$ (capital K_t demanded by firms equals aggregate savings Na_{t-1} supplied by households).
- Labor market: $L_t = N$ (labor L_t demanded by firms equals exogenously supplied labor by households).

- Goods market: $Y_t + (1 - \delta)K_t = N(c_{y,t} + c_{o,t} + a_t)$ (aggregate output & undepreciated capital equals the amount of goods consumed by young and old each period and investment by the young).

1.2.4 Analytical results

The household's first-order conditions are unchanged from earlier, but we have to be more careful with the timing. The optimal savings rate in t is now given by

$$s_t = \left[1 + \beta^{-\frac{1}{\gamma}} (1 + r_{t+1})^{1 - \frac{1}{\gamma}} \right]^{-1} \quad (3)$$

It is important to note that the savings rate s_t in period t depends on the return on savings r_{t+1} realized in $t + 1$, which in turn depends on s_t . Because of the perfect foresight assumption, rational households know the future path of interest rates after a shock hits the economy.

As you can guess from (3), the expression for the savings rate does not have an analytical solution in general. We instead solve for the transition dynamics using the simplifying assumption of log preferences with $\gamma = 1$, as then the savings rate becomes

$$s_t = \frac{\beta}{1 + \beta}$$

In this case, the savings rate is constant across time and only depends on parameters. We therefore don't need to know r_{t+1} to know how much young households want to save. This is of course the consequence of log preferences where income and substitution effects cancel out, and the household's optimal savings choice does not depend on r .

We relax the assumption of log preferences in the optional exercises at the end of this lecture. This requires an additional root-finding step in each simulation period.

1.2.5 Transition dynamics

Consider the following sequence of events:

1. The economy is in the steady state implied by the original parameters.
2. The economy is hit by an unexpected *permanent* drop in TFP z of 10%. All households understand that z is going to remain at this lower level forever and adjust their choices accordingly.

Transition path

We can compute household choices and aggregates along the transition path towards the new steady state implied by the permanently lower TFP level using the following algorithm. We maintain the assumption of log preferences so that the savings rate s is constant along the transition path.

- Period $t = 0$: The economy is in the steady state with $K_0 = K^*$, $w_0 = w^*$, $a_0 = a^*$, etc.
- Period $t \geq 1$:
 - The capital stock is predetermined from the previous period, $K_t = Na_{t-1}$
 - Production takes place:
 - * Young households earn $w_t = (1 - \alpha)z_t(K_t/L)^\alpha$
 - * Old households earn gross asset returns $(1 + r_t)a_{t-1}$ with $r_t = \alpha z_t(K_t/L)^{\alpha-1} - \delta$
 - Consumption and savings:
 - * Young households save $a_t = sw_t$ and consume $c_{y,t} = (1 - s)w_t$
 - * Old households consume $c_{o,t} = (1 + r_t)a_{t-1}$

1.2.6 Numerical implementation

We first define a data class called `Simulation` which stores the time series for each simulated variable in the economy. Note that now the data type is declared to be `np.ndarray` since these attributes are going to be arrays.

```
[13]: import numpy as np

@dataclass
class Simulation:
    c_y: np.ndarray = None      # Time series for consumption when young
    c_o: np.ndarray = None      # Time series for consumption when old
    a: np.ndarray = None        # Time series for savings when young
    s: np.ndarray = None        # Time series for savings rate when young
    r: np.ndarray = None        # Time series for interest rate (return on capital)
    w: np.ndarray = None        # Time series for wages
    K: np.ndarray = None        # Time series for aggregate capital stock
    Y: np.ndarray = None        # Time series for aggregate output
    z: np.ndarray = None        # Time series for TFP
```

Next, we compute the initial steady-state equilibrium which is the starting point of the simulation:

```
[14]: # Parameter instance with risk aversion gamma=1
par = Parameters(gamma=1)

# Compute equilibrium at original TFP level
eq_init = compute_steady_state(par)

# Print initial equilibrium
print_steady_state(eq_init)
```

Steady-state equilibrium:

```
Households:
  c_y = 0.16717
  c_o = 0.12167
  a = 0.04912
Firms:
  K = 0.04912
  L = 1.00000
  Y = 0.33796
Prices:
  r = 1.47669
  w = 0.21630
Market clearing:
  Capital market: -1.98591e-14
  Goods market: 4.92384e-14
```

We now implement the function `simulate_olg()` which takes the initial equilibrium as a starting point, assumes that TFP z permanently drops to the new value z_{new} , and simulates the transition dynamics for T periods.

We use the helper function `initialize_sim()` from the module `lecture07_olg` which initializes the simulated time series to the required array sizes and sets the first element of each series to the corresponding initial equilibrium value.

```
[15]: import copy
from lecture07_olg import initialize_sim

def simulate_olg(z_new, eq: SteadyState, T = 10):
    """
    Simulate the transition dynamics of the overlapping generations model.
```

```

Parameters
-----
z_new : float
    New level of TFP after the shock.
eq : SteadyState
    Initial steady-state equilibrium before the shock.
T : int
    Number of periods to simulate.

Returns
-----
sim : Simulation
    Simulation object containing the time series for each variable.
"""

# Retrieve parameter object attached to steady-state equilibrium
par = eq.par

# The following code only works for log utility
if par.gamma != 1:
    raise ValueError('Simulation only implemented for log utility')

# Initialize simulation instance and allocate arrays
sim = initialize_sim(T, eq)

# TFP is assumed to be at new level for all remaining periods
sim.z[1:] = z_new

# Copy parameters to avoid changing the original instance
par_ = copy.copy(par)

# Savings rate is independent of r for gamma = 1 and constant over time
s = par.beta / (1 + par.beta)
sim.s[:] = s

for t in range(1, T+1):
    # Update TFP with current value
    par_.z = sim.z[t]

    # Capital stock is predetermined by savings of old in previous period
    sim.K[t] = sim.a[t-1] * par.N

    # Prices given predetermined capital stock and current z
    sim.r[t], sim.w[t] = compute_prices(sim.K[t] / par.N, par_)

    # Savings by the young
    sim.a[t] = s * sim.w[t]
    # Consumption by the young
    sim.c_y[t] = (1-s) * sim.w[t]
    # Consumption by the old
    sim.c_o[t] = (1 + sim.r[t]) * sim.a[t-1]
    # Aggregate output
    sim.Y[t] = sim.z[t] * sim.K[t]**par.alpha * par.N**(1-par.alpha)

    # Verify that goods market clearing holds
    demand = par.N * (sim.c_y[t] + sim.c_o[t] + sim.a[t])
    supply = sim.Y[t] + (1 - par.delta) * sim.K[t]
    assert abs(demand - supply) < 1.0e-8

return sim

```

Using `simulate_olg()`, we simulate $T = 10$ periods (in addition to the initial period $t = 0$):

```
[16]: # Number of periods to simulate
T = 20

# New TFP level (10% drop from steady state)
z_new = 0.9 * par.z

# Perform simulation
sim = simulate_olg(z_new, eq_init, T=T)
```

As we'll see shortly, the economy converges very quickly to the new steady state implied by $z = 0.9$, which we can compute explicitly:

```
[17]: # Compute new steady state using new TFP level
eq_new = compute_steady_state(par=Parameters(gamma=par.gamma, z=z_new))

# Print new steady state (transition end point)
print_steady_state(eq_new)
```

Steady-state equilibrium:

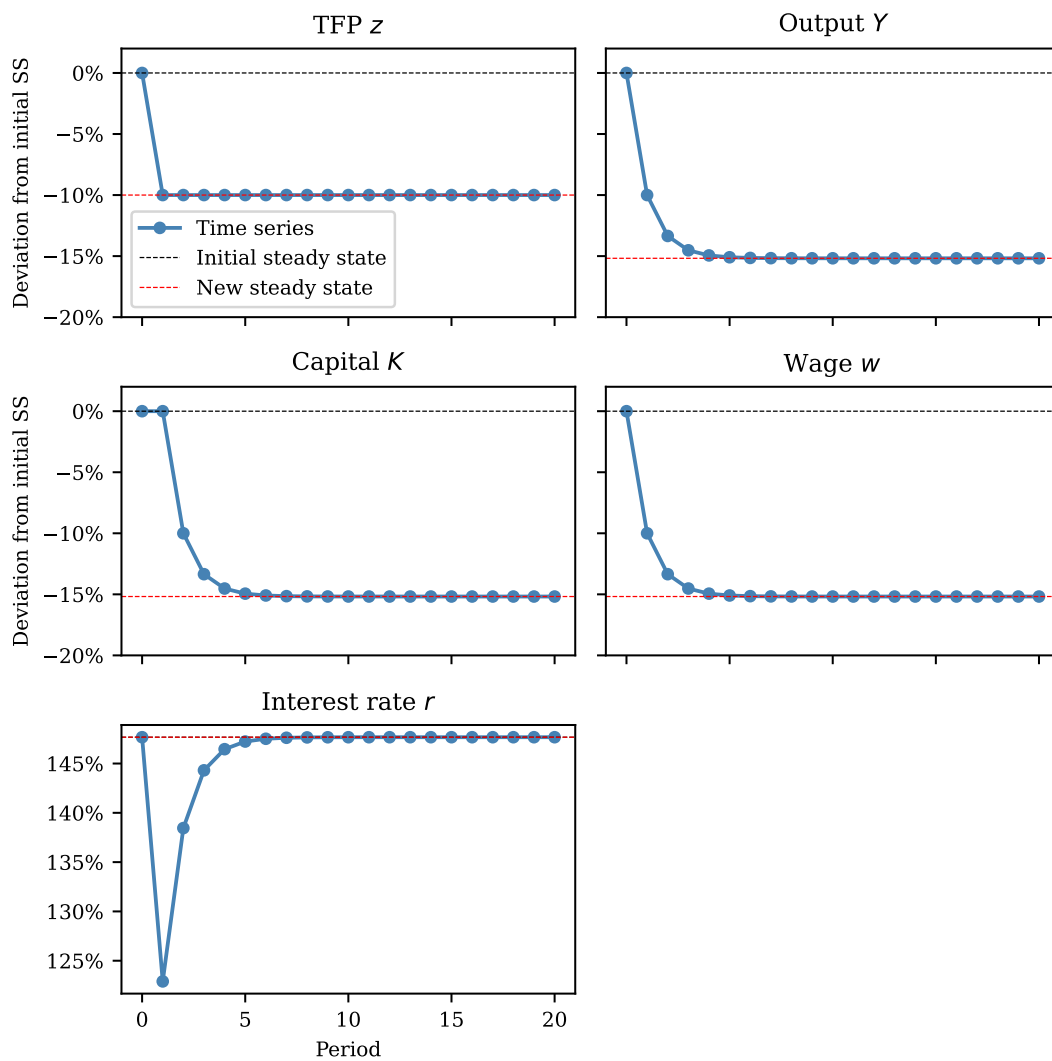
```
Households:
  c_y = 0.14180
  c_o = 0.10320
  a = 0.04167
Firms:
  K = 0.04167
  L = 1.00000
  Y = 0.28666
Prices:
  r = 1.47669
  w = 0.18346
Market clearing:
  Capital market: -1.22541e-14
  Goods market: 3.04201e-14
```

As you can see, the new steady state is characterized by a lower capital stock K , lower output Y , and lower wages w . However, the interest rate is identical in both states. The reason is that the steady-state interest rate in this model only depends on the parameters β , γ , and α , but not on z (this is not overly surprising since z changes the scale of the economy, but this should not affect relative quantities such as the interest rate).

Lastly, we use the function `plot_simulation()` implemented in the `lecture07_olg` module to plot selected impulse response functions (IRFs) along the transition path. We do this in terms of relative deviations from the initial steady state (a standard way to display IRFs in macroeconomics), except for the interest rate which is plotted in absolute deviations from the initial steady state.

```
[18]: from lecture07_olg import plot_simulation

# Plot impulse responses for selected variables
_ = plot_simulation(eq_init, sim, eq_new)
```



As the figure shows, the economy very quickly converges to the new steady state within a few periods.

Your turn.

As a robustness check, verify that the simulated trajectories are *constant* if the new TFP level is unchanged from the initial TFP level.