Семинар 15

1. Модель Леонтьева

1.1. Теорема Перрона-Фробениуса

Матрица $n \times m$ называется неотрицательной, если каждый её элемент $a_{ij} \geqslant 0, \ \forall i, j$. Для краткости обозначим это как $A \geqslant 0$.

Матрица $n \times n$ называется несократимой, если $A + A^2 + A^3 + \ldots \gg 0$, где $\gg$ означает положительность всех элементов матрицы.

Нам известно, что такое собственное значение λ и собственный вектор v матрицы A:

$$Av = \lambda v$$

Данные собственные значения также называются **правыми**. Значит, есть и **левые**:

$$A^{\mathsf{T}} w = \lambda w$$
$$w^{\mathsf{T}} A = \lambda w^{\mathsf{T}}$$

```
import numpy as np
from numpy.linalg import eig
import scipy as sp
import matplotlib.pyplot as plt
```

```
A = np.array([[3, 2], [1, 4]])

# Compute eigenvalues and right eigenvectors
λ, v = eig(A)

# Compute eigenvalues and left eigenvectors
λ, w = eig(A.T)

# Keep 5 decimals
np.set_printoptions(precision=5)

print(f"The eigenvalues of A are:\n {λ}\n")
print(f"The corresponding right eigenvectors are: \n {v[:, 0]}␣
  ↪and {-v[:, 1]}\n")
print(f"The corresponding left eigenvectors are: \n {w[:, 0]}␣
  ↪and {-w[:, 1]}\n")
```

```
The eigenvalues of A are:
 [2. 5.]

The corresponding right eigenvectors are:
 [-0.89443  0.44721] and [0.70711 0.70711]

The corresponding left eigenvectors are:
 [-0.70711  0.70711] and [0.44721 0.89443]
```

```python
eigenvals, ε, e = sp.linalg.eig(A, left=True)

print(f"The eigenvalues of A are:\n {eigenvals.real}\n")
print(f"The corresponding right eigenvectors are: \n {e[:, 0]}␣
  ↪and {-e[:, 1]}\n")
print(f"The corresponding left eigenvectors are: \n {ε[:, 0]}␣
  ↪and {-ε[:, 1]}\n")
```

```
The eigenvalues of A are:
 [2. 5.]

The corresponding right eigenvectors are:
 [-0.89443  0.44721] and [0.70711 0.70711]

The corresponding left eigenvectors are:
 [-0.70711  0.70711] and [0.44721 0.89443]
```

1.1.1. Теорема

Если матрица $A \geqslant 0$, то

1. доминирующее собственное значение (максимальное по абсолютному значению) $r(A) \in \mathbb{R}$ и $r(A) > 0$;
2. для всех собственных значений матрицы $|\lambda_i| \leqslant r(A)$;
3. существует неотрицательный и ненулевой вектор w такой, что $Aw = r(A)w$.

Если матрица несократима, то

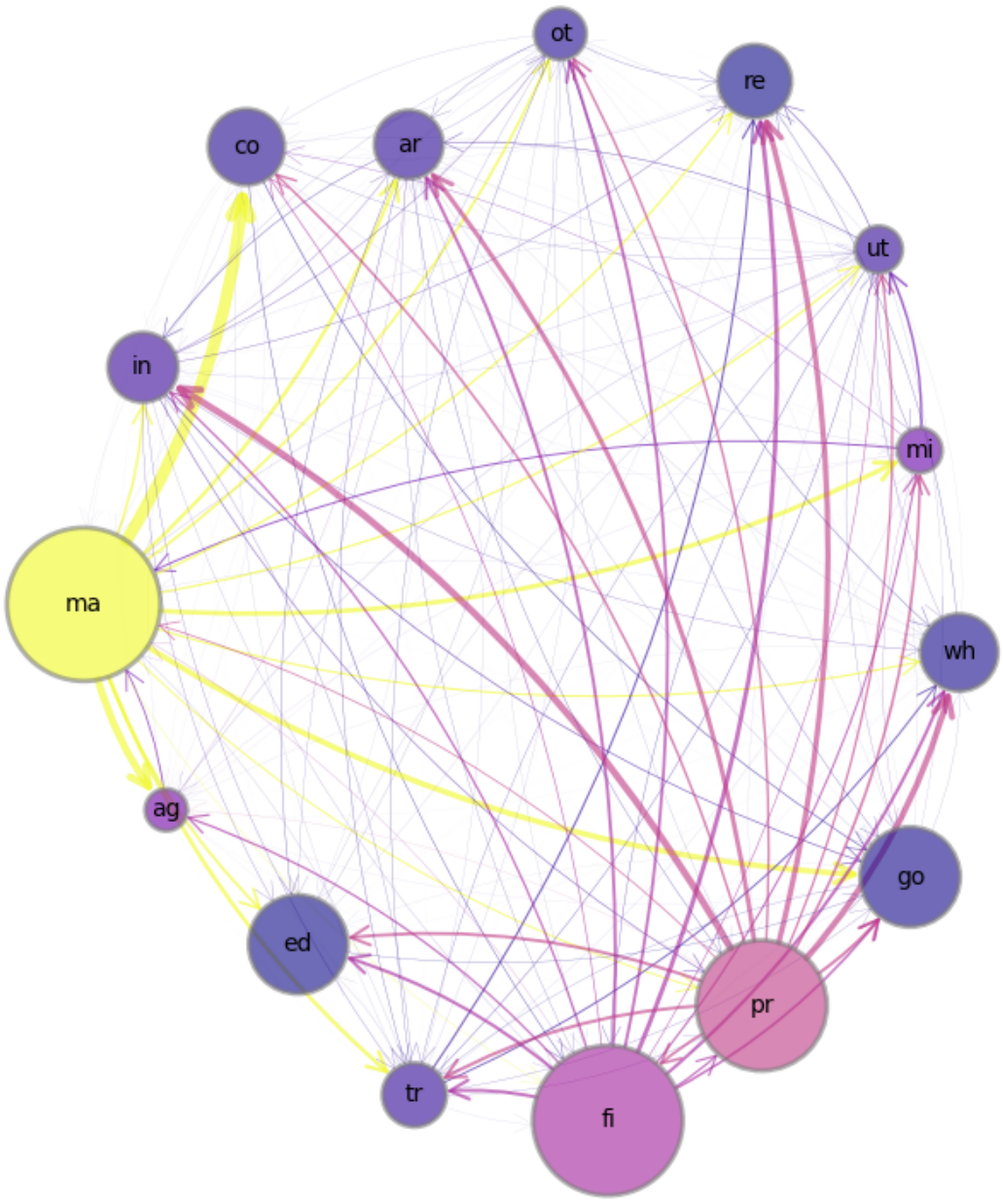
4. вектор w строго положителен;
5. вектор w единственен.

Если матрица $A \geqslant 0$ примитивна, то есть $K \in \mathbb{N} : A^k > 0, \ \forall k \geqslant N$, то

6. для всех собственных значений матрицы $|\lambda_i| < r(A), \ \forall \lambda_i \neq r(A)$;
7. если левый w и правый v собственные вектора, соответствующие $r(A)$ нормированы так, что $(w, v) = 1$, то $r(A)^{-m} A^m \to_{m\to\infty} vw^{\mathsf{T}}$.

1.2. Модель Леонтьева

Label	Sector	Label	Sector	Label	Sector
ag	Agriculture	wh	Wholesale	pr	Professional Services
mi	Mining	re	Retail	ed	Education & Health
ut	Utilities	tr	Transportation	ar	Arts & Entertainment
co	Construction	in	Information	ot	Other Services (exc govt)
ma	Manufacturing	fi	Finance	go	Government

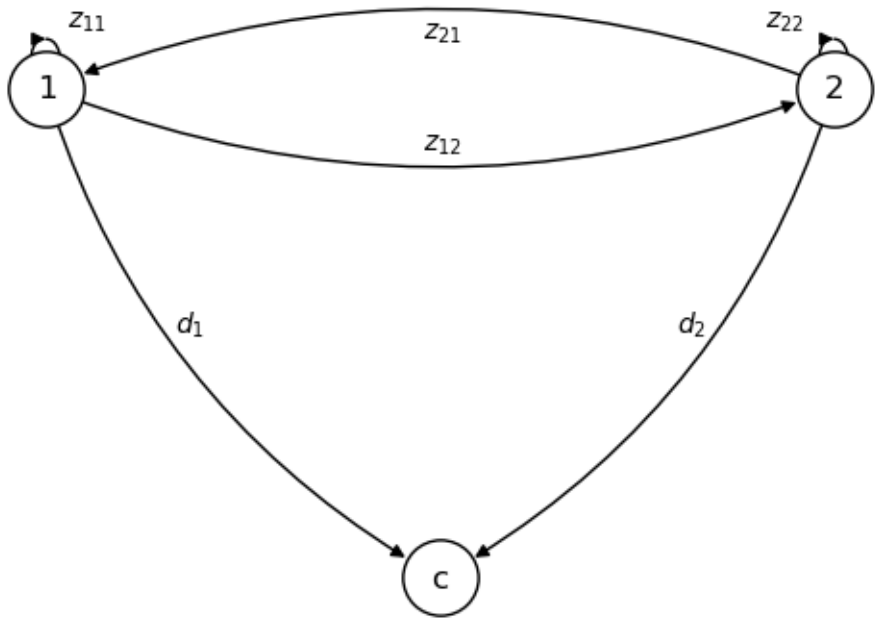


Пусть

- x_0 величина экзогенного ресурса (труда);
- $x_j, j = 1, \dots n$ выпуск-брутто товара j;
- $d_j, j = 1, \dots n$ итоговое потребление товара j;
- z_{ij} количество товара i, являющееся сырьем для производства товара j для $i = 1, \dots n$, $j = 1, \dots n$;
- z_{0j} количество труда, направленное на производство товара j;
- a_{ij} количество товара i, необходимое для производства единицы товара j, $i = 0, \dots, n, j = 1, \dots n$;
- $w > 0$ экзогенная ставка зарплаты в долларах за единицу труда;
- $p = n \times 1$ — вектор цен на товары $i = 1, \dots, n$.

Производственная технология для товара $j \in \{1, \dots, n\}$ описывается функцией Леонтьева

$$x_j = \min_{i \in \{0,\dots,n\}} \left(\frac{z_{ij}}{a_{ij}} \right)$$



Допустимые решения должны удовлетворять следующим условиям

$$\begin{aligned} (1 - a_{11})x_1 - a_{22}x_2 &\geqslant d_1 \\ -a_{21}x_1 + (1 - a_{22})x_2 &\geqslant d_2 \\ a_{01}x_1 + a_{02}x_2 &\leqslant x_0 \end{aligned}$$

или

$$\begin{aligned} (I - A)x &\geqslant d \\ a_0^\top x &\leqslant x_0 \end{aligned}$$

Решение имеет вид

$$\tilde{x} = \underbrace{(I - A)^{-1}}_{L} d$$

$\tilde{x}$ будет положительным вектором, если выполняются условия Хоукинса-Саймона

- $\det(I - A) > 0$;
- $(I - A)_{ii} > 0$.

Из второго уравнения имеем

$$a_0^\top \tilde{x} = A_0 d = x_0,$$

где $A_0 = a_0 L$. Это выражение дает соотношение между величиной вложенного труда и наборами итогового потребления. Элементы A_0 — объемы труда для производства соответствующих товаров.

1.2.1. Цены

[Dorfman *et al.*, 1958] утверждают, что относительные цены должны удовлетворять

$$p_1 = a_{11} p_1 + a_{21} p_2 + a_{01} w$$
$$p_2 = a_{12} p_1 + a_{22} p_2 + a_{02} w$$

или

$$p = A^\top p + a_0 w$$

то есть цена итогового товара равна совокупным издержкам производства: стоимости промежуточных товаров и вложенного труда. Это уравнение можно переписать в виде

$$(I - A^\top) p = a_0 w$$

что влечет

$$\tilde{p} = (I - A^\top)^{-1} a_0 w = L a_0 w$$

1.2.2. Линейное программирование

Обратите внимание, что уравнения для $\tilde{x}$ и $\tilde{p}$ формируют прямую и дуальную задачи!

Прямая задача имеет вид

$$\min_x \underbrace{w a_0^\top x}_{x_0}$$
$$(I - A) x \geqslant d$$

то есть мы минимизируем итоговую стоимость производства заданного потребительского набора.

Дуальная задача имеет вид

$$\max_p p^\top d$$
$$(I - A)^\top p \leqslant w a_0$$

то есть мы максимизируем стоимость заданного потребительского набора, требуя. чтобы цены покрывали затраты на производство.

Согласно теореме строгой дуальности

$$w a_0^\top x^* = p^* d$$

1.2.3. Шоки спроса

Рассмотрим шок спроса Δd такой, что d_0 превращается в $d_1 = d_0 + \Delta d$. Брутто-выпуск меняется с $x_0 = Ld_0$ на $x_1 = Ld_1$.

Если $r(A) < 1$, то согласно Лемме рядов Неймана существует решение и

$$\Delta x = L\Delta d = \Delta d + A(\Delta d) + A^2(\Delta d) + \cdots$$

Это значит, что L_{ij} показывает влияние единичного шока спроса d_j на выпуск товара i.

1.3. Эксперимент

```
A = np.array([[0.1, 40], [0.01, 0]])
d = np.array([50, 2]).reshape((2, 1))
```

```
I = np.identity(2)
B = I - A
B
```

```
array([[ 9.e-01, -4.e+01],
       [-1.e-02,  1.e+00]])
```

```
np.linalg.det(B) > 0  # checking Hawkins-Simon conditions
```

```
np.True_
```

```
L = np.linalg.inv(B)  # obtaining Leontief inverse matrix
L
```

```
array([[2.0e+00, 8.0e+01],
       [2.0e-02, 1.8e+00]])
```

```
x = L @ d  # solving for gross output
x
```

```
array([[260. ],
       [  4.6]])
```

```
# Пусть мы имеем следующие требования к величине используемого␣
  ↪труда
a0 = np.array([4, 100])
A0 = a0 @ L
A0
```

```
array([ 10., 500.])
```

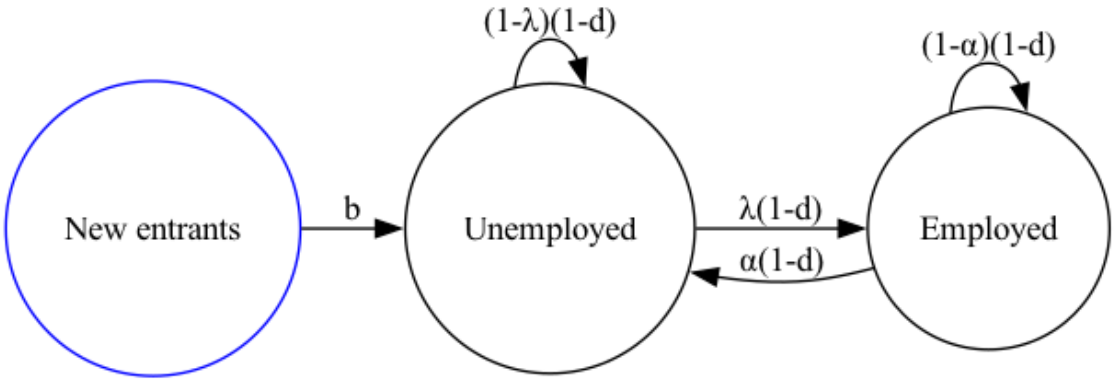
2. Модель рынка труда

Эта модель часто называет **моделью пруда**, так как в ней есть два «пруда» (пула) работников:

- занятые,
- безработные, но ищущие работу.

Потоки между «прудами»:

- работники выбывают с рынка со скоростью d;
- новые работники входят на рынок со скоростью b;
- занятые увольняются со скоростью α;
- безработные находят работу со скоростью λ.



Пусть e_t и u_t — численность занятых и безработных в момент времени t. Совокупная численность населения $n_t = e_t + u_t$.

Численность меняется в соответствии с уравнениями

$$\begin{aligned} u_{t+1} &= (1-d)(1-\lambda)u_t + \alpha(1-d)e_t + bn_t \\ &= ((1-d)(1-\lambda) + b)u_t + (\alpha(1-d) + b)e_t \\ e_{t+1} &= (1-d)\lambda u_t + (1-\alpha)(1-d)e_t \end{aligned}$$

или в матричном представлении

$$\underbrace{\begin{bmatrix} u_{t+1} \\ e_{t+1} \end{bmatrix}}_{x_{t+1}} = \underbrace{\begin{bmatrix} (1-d)(1-\lambda)+b & \alpha(1-d)+b \\ (1-d)\lambda & (1-\alpha)(1-d) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u_t \\ e_t \end{bmatrix}}_{x_t}$$

Пусть в момент $t = 0$ мы имеем $x_0 = \begin{bmatrix} u_0 & e_0 \end{bmatrix}^\mathsf{T}$. Тогда $x_t = A^t x_0$.

Нетрудно увидеть, что

$$\begin{aligned} n_{t+1} &= u_{t+1} + e_{t+1} \\ &= \mathbb{1}^\mathsf{T} x_{t+1} \\ &= \mathbb{1}^\mathsf{T} A x_t \\ &= (1 + b - d)(u_t + e_t) \\ &= (1 + b - d)n_t. \end{aligned}$$

где $\mathbb{1}$ — столбец из единиц.

2.1. Теорема Перрона-Фробениуса

Так как матрица A, введенная выше, неотрицательна и несократима, то

- существуют левый ψ и правый ϕ собственные вектора

$$A\phi = r(A)\phi$$
$$\psi A = r(A)\psi$$

- если A положительна, то $(\psi, \phi) = \psi\phi = 1$ и $r(A)^{-t}A^t \to \phi\psi$.

Так как спектральный радиус $r(A)$ ограничен суммами элементов столбцов

$$\min_j \operatorname{colsum}_j(A) \le r(A) \le \max_j \operatorname{colsum}_j(A)$$

и поскольку эти суммы для нашей матрицы одинаковы, то

$$r(A) = 1 + b - d$$

Из этого также следует, что $\mathbb{1}$ — левый собственный вектор.

Отсюда можем найти вектор $\psi = \bar{x}$

$$\bar{u} = \frac{b + \alpha(1-d)}{b + (\alpha+\lambda)(1-d)}$$
$$\bar{e} = \frac{\lambda(1-d)}{b + (\alpha+\lambda)(1-d)}$$

Этот вектор играет важную роль в поиске долгосрочного равновесия.

2.2. Уровни безработицы

Теорема Перрона-Фробениуса утверждает, что

$$r(A)^{-t}A^t \approx \bar{x}\mathbb{1}^\top = \begin{bmatrix} \bar{u} & \bar{u} \\ \bar{e} & \bar{e} \end{bmatrix}$$

Поэтому для любого $x_0 = (u_0, e_0)^\top$ при достатчно большом t мы имеем

$$\begin{aligned} x_t = A^t x_0 &\approx r(A)^t \begin{bmatrix} \bar{u} & \bar{u} \\ \bar{e} & \bar{e} \end{bmatrix} \begin{bmatrix} u_0 \\ e_0 \end{bmatrix} = \\ &= (1+g)^t (u_0 + e_0) \begin{bmatrix} \bar{u} \\ \bar{e} \end{bmatrix} = \\ &= (1+g)^t n_0 \bar{x} = \\ &= n_t \bar{x} \end{aligned}$$

Нормы занятости и безработицы равны $r_t \coloneqq \dfrac{x_t}{n_t} = A^t \dfrac{x_0}{n_t} \to \bar{x}$. Динамика норм равна

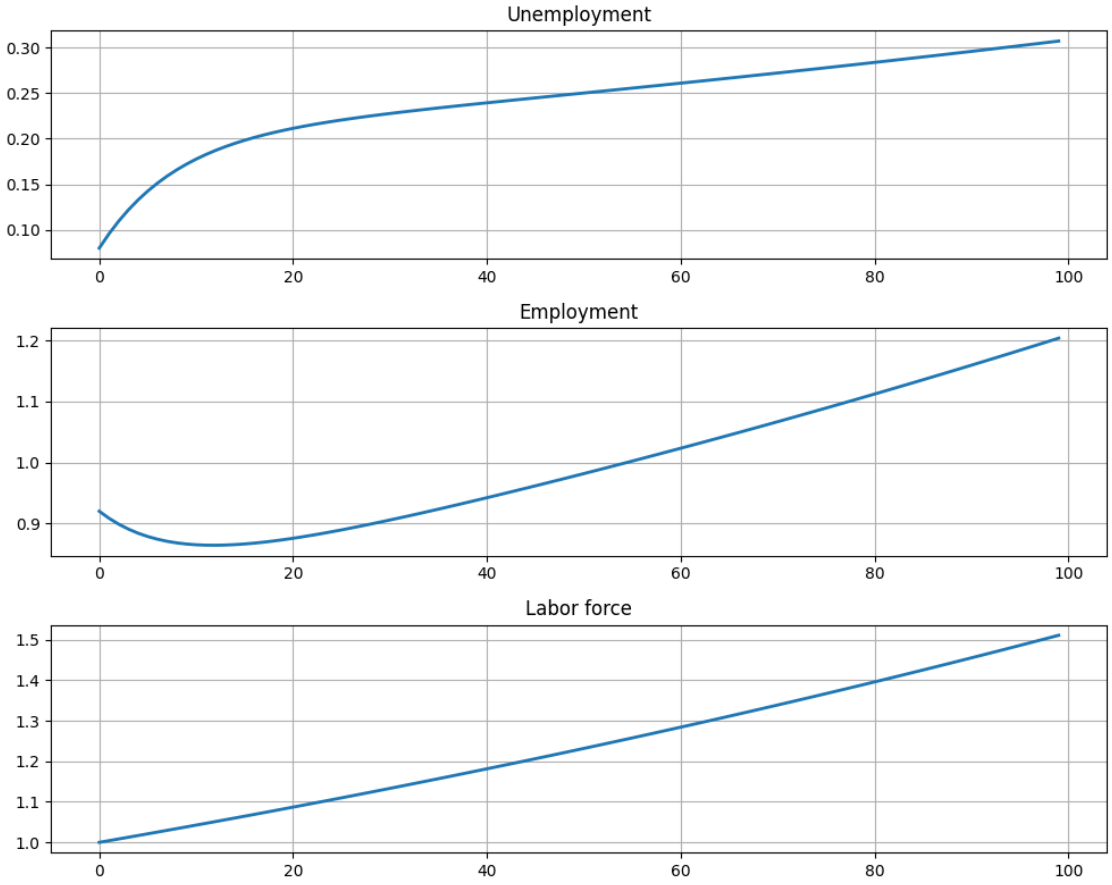
$$r_{t+1} = \frac{x_{t+1}}{n_{t+1}} = \frac{x_{t+1}}{(1+g)n_t} = \frac{Ax_t}{(1+g)n_t} = \hat{A}\frac{x_t}{n_t} = \hat{A}r_t$$

2.3. Эксперимент

```
class LakeModel:
    """
    Solves the lake model and computes dynamics of the unemployment stocks and
    rates.

    Parameters:
    ------------
    λ : scalar
        The job finding rate for currently unemployed workers
    α : scalar
        The dismissal rate for currently employed workers
    b : scalar
        Entry rate into the labor force
    d : scalar
        Exit rate from the labor force

    """

    def __init__(self, λ=0.1, α=0.013, b=0.0124, d=0.00822):
        self.λ, self.α, self.b, self.d = λ, α, b, d

        λ, α, b, d = self.λ, self.α, self.b, self.d
        self.g = b - d
        g = self.g

        self.A = np.array(
            [[(1 - d) * (1 - λ) + b, α * (1 - d) + b], [(1 - d) * λ, (1 - α) * (1 - d)]]
        )

        self.U = (1 + g - (1 - d) * (1 - α)) / (1 + g - (1 - d) * (1 - α) + (1 - d) * λ)
        self.E = 1 - self.U

    def simulate_path(self, x0, T=1000):
        """
        Simulates the sequence of employment and unemployment

        Parameters
        ----------
        x0 : array
            Contains initial values (u0,e0)
        T : int
            Number of periods to simulate

        Returns
        ----------
        x : iterator
```

```
            Contains sequence of employment and unemployment rates
        """
        x0 = np.atleast_1d(x0)  # Recast as array just in case
        x_ts = np.zeros((2, T))
        x_ts[:, 0] = x0
        for t in range(1, T):
            x_ts[:, t] = self.A @ x_ts[:, t - 1]
        return x_ts
```

```
e_0 = 0.92  # Initial employment
u_0 = 1 - e_0  # Initial unemployment, given initial n_0 = 1
T = 100  # Simulation length

lm = LakeModel()

x_0 = (u_0, e_0)
x_path = lm.simulate_path(x_0, T)

fig, axes = plt.subplots(3, 1, figsize=(10, 8))

axes[0].plot(x_path[0, :], lw=2)
axes[0].set_title("Unemployment")

axes[1].plot(x_path[1, :], lw=2)
axes[1].set_title("Employment")

axes[2].plot(x_path.sum(0), lw=2)
axes[2].set_title("Labor force")

for ax in axes:
    ax.grid()

plt.tight_layout()
plt.show()
```

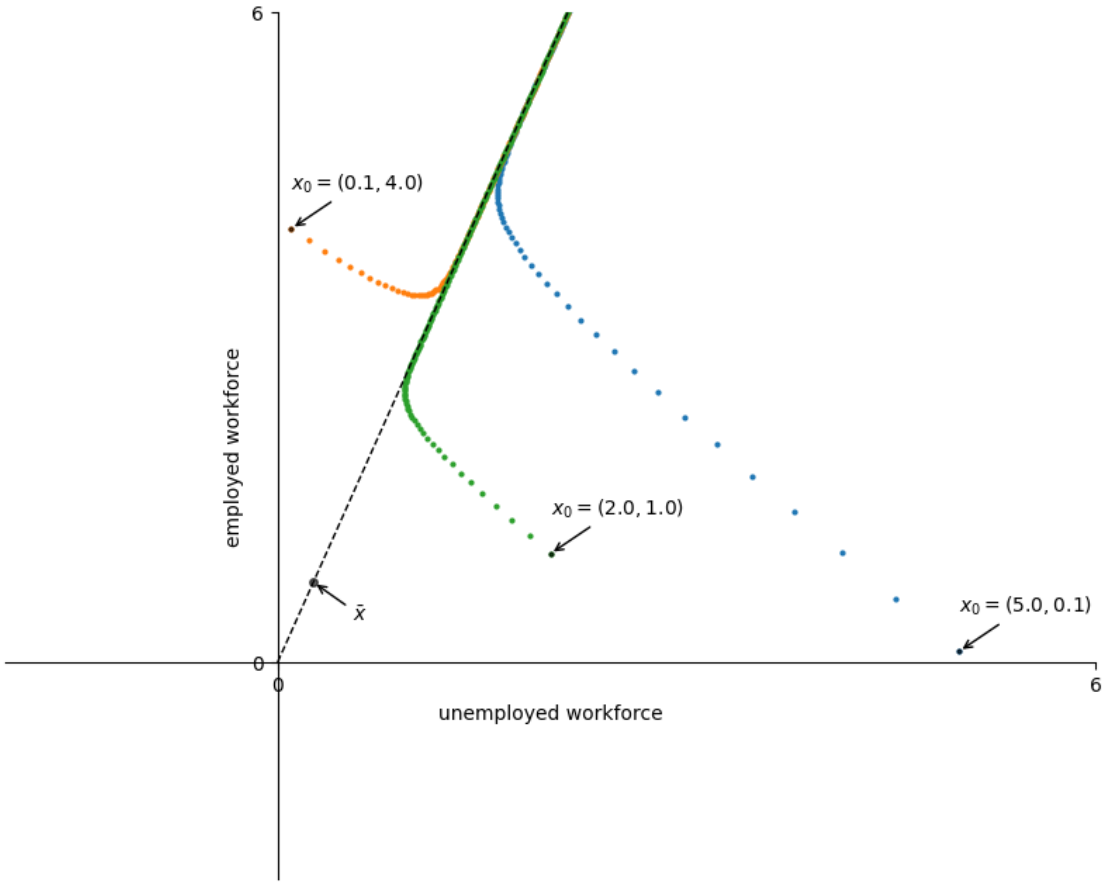



```
def plot_time_paths(lm, x0=None, T=1000, ax=None):
    """
    Plots the simulated time series.

    Parameters
    ----------
    lm : class
        Lake Model
    x0 : array
        Contains some different initial values.
    T : int
        Number of periods to simulate
    """

    if x0 is None:
        x0 = np.array([[5.0, 0.1]])

    U, E = lm.U, lm.E

    x0 = np.atleast_2d(x0)

    if ax is None:
        fig, ax = plt.subplots(figsize=(10, 8))
```

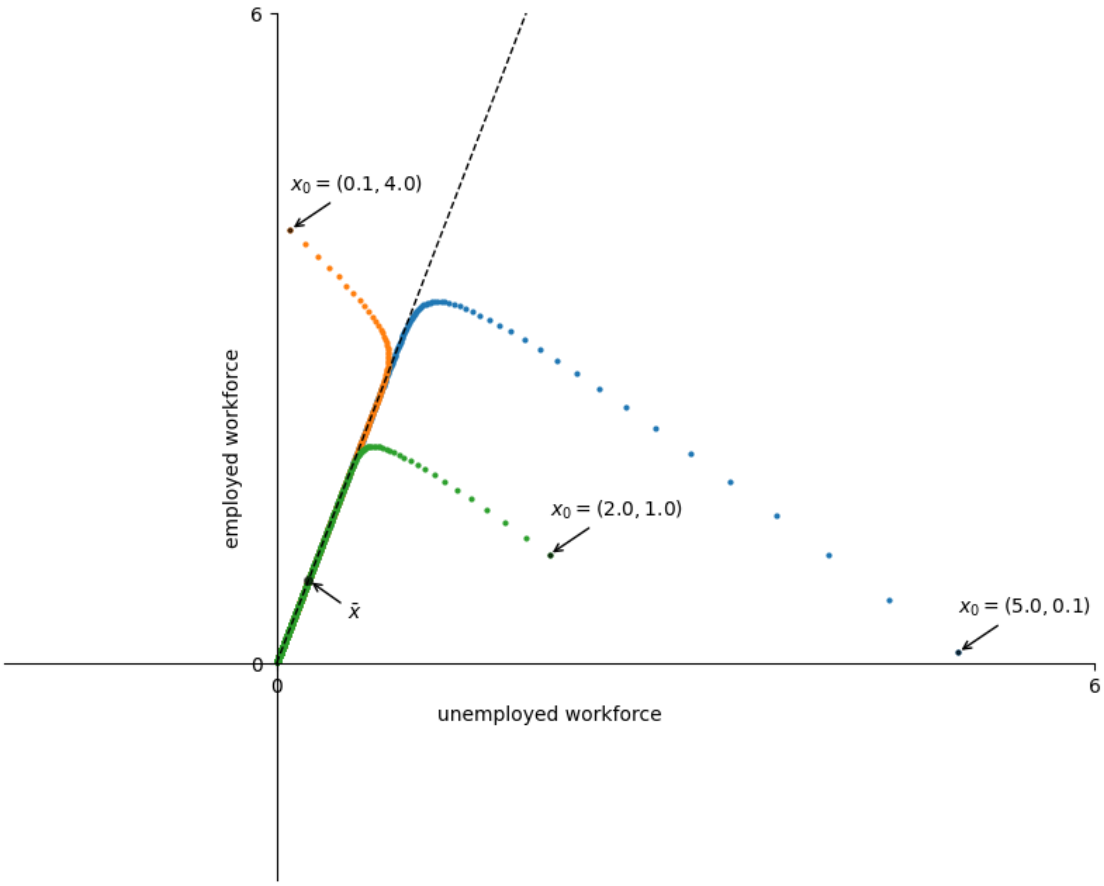
```
        # Plot line D
        s = 10
        ax.plot([0, s * U], [0, s * E], "k--", lw=1, label="set␣
↪$D$")

    # Set the axes through the origin
    for spine in ["left", "bottom"]:
        ax.spines[spine].set_position("zero")
    for spine in ["right", "top"]:
        ax.spines[spine].set_color("none")

    ax.set_xlim(-2, 6)
    ax.set_ylim(-2, 6)
    ax.set_xlabel("unemployed workforce")
    ax.set_ylabel("employed workforce")
    ax.set_xticks((0, 6))
    ax.set_yticks((0, 6))

    # Plot time series
    for x in x0:
        x_ts = lm.simulate_path(x0=x)

        ax.scatter(
            x_ts[0, :],
            x_ts[1, :],
            s=4,
        )

        u0, e0 = x
        ax.plot([u0], [e0], "ko", ms=2, alpha=0.6)
        ax.annotate(
            f"$x_0 = ({u0},{e0})$",
            xy=(u0, e0),
            xycoords="data",
            xytext=(0, 20),
            textcoords="offset points",
            arrowprops=dict(arrowstyle="->"),
        )

    ax.plot([U], [E], "ko", ms=4, alpha=0.6)
    ax.annotate(
        r"$\bar{x}$",
        xy=(U, E),
        xycoords="data",
        xytext=(20, -20),
        textcoords="offset points",
        arrowprops=dict(arrowstyle="->"),
    )
```

```
    if ax is None:
        plt.show()
```

```
lm = LakeModel(α=0.01, λ=0.1, d=0.02, b=0.025)
x0 = ((5.0, 0.1), (0.1, 4.0), (2.0, 1.0))
plot_time_paths(lm, x0=x0)
```




```
lm = LakeModel(α=0.01, λ=0.1, d=0.025, b=0.02)
plot_time_paths(lm, x0=x0)
```




```
lm = LakeModel()
e_0 = 0.92  # Initial employment
u_0 = 1 - e_0  # Initial unemployment, given initial n_0 = 1

lm = LakeModel()
T = 100  # Simulation length

x_0 = (u_0, e_0)

x_path = lm.simulate_path(x_0, T)

rate_path = x_path / x_path.sum(0)

fig, axes = plt.subplots(2, 1, figsize=(10, 8))

# Plot steady ū and ē
axes[0].hlines(lm.U, 0, T, "r", "--", lw=2, label="ū")
axes[1].hlines(lm.E, 0, T, "r", "--", lw=2, label="ē")

titles = ["Unemployment rate", "Employment rate"]
locations = ["lower right", "upper right"]

# Plot unemployment rate and employment rate
```

```
for i, ax in enumerate(axes):
    ax.plot(rate_path[i, :], lw=2, alpha=0.6)
    ax.set_title(titles[i])
    ax.grid()
    ax.legend(loc=locations[i])


plt.tight_layout()
plt.show()
```