Семинар 7

1. Марковские цепи

1.1. Определение

Марковские цепи представляют собой важный класс стохастических **процессов**, который

- прост, гибок и имеет хорошую теоретическую основу;
- полезен для описания других стохастических процессов;
- важен для вычислительной экономики.

Стохастическая или **Марковская** матрица — матрица $P_{n\times n}$ такая, что

- все $p_{ij} \geqslant 0$;
- сумма элементов в каждой строке равна 1: $\sum_j p_{ij} = 1,\ \forall i$.

Нетрудно показать, что если P и Q — стохастические, то PQ — тоже. Действительно, рассмотрим **первую** строку матрицы PQ. На первом месте будет стоять $(p_1, Q_1) = \sum_i p_{1i}q_{i1}$, на втором — (p_1, Q_2), и так далее. Найдем сумму элементов первой строки PQ

$$\sum_i (p_1, Q_i) = \sum_i \sum_j p_{1j}q_{ji} = \star$$

С очевидностью имеем

$$\star = \sum_j p_{1j} \sum_i q_{ji} = \star$$

Из определения стохастической матрицы имеем, что $\sum_i q_{ji} = 1$. Тогда

$$\star = \sum_j p_{1j} = 1$$

Очевидно, что P^k — стохастическая.

Рассмотрим некое множество S из n элементов $\{x_1, x_2, \dots, x_n\}$. Элементы множества называются **состояниями**, а само множество — **пространством состояний**.

Марковской цепью на множестве S называется последовательность случайных величин $\{X_t\}$ на множестве S, обладающая **марковским** свойством

$$\Pr\{X_{t+1} = y \mid X_t, X_{t-1}, \dots\} = \Pr\{X_{t+1} = y \mid X_t\}, \quad y \in S$$

То есть для прогнозирования будущих значений цепи достаточно знать текущее состояние.

Динамика марковской цепи полностью определяется набором значений

$$P(x, y) = \Pr\{X_{t+1} = y \mid X_t = x\}, \quad x, y \in S$$

Понятно, что

$$\sum_{i=1}^{n} P(x, y_i) = 1$$

то есть $P(\cdot, \cdot)$ представима в виде стохастической матрицы

$$P = (p_{ij}) = (P(x_i, y_j))$$

Также нужно ввести понятие **предельного распределения** — набора *безусловных* вероятностей

$$\psi_t(x) := \Pr\{X_t = x\}$$
$$\psi_t := [\psi_t(x_1), \psi_t(x_2), \dots, \psi_t(x_n)]$$

Для генерации марковской цепи надо

1. получить начальное значение X_0 из начального предельного распределения ψ_0;
2. для $t = 0, 1, \dots$ выбрать значение X_{t+1} из распределения $P(X_t, \cdot)$, определяемого X_t-й строкой матрицы P.

1.1.1. Пример

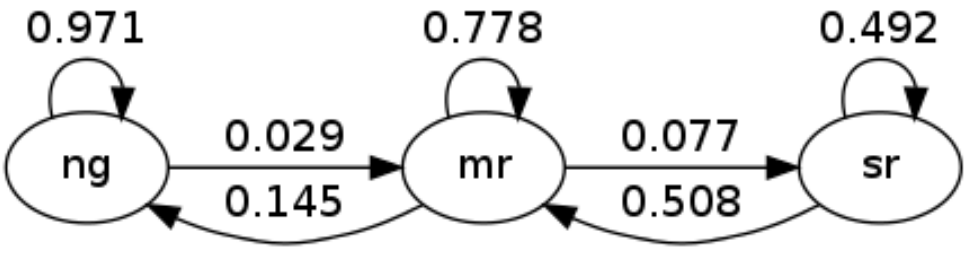
Из данных для уровня безработицы в США Гамильтон оценил матрицу P следующим образом

$$P = \begin{pmatrix} 0.971 & 0.029 & 0 \\ 0.145 & 0.778 & 0.077 \\ 0 & 0.508 & 0.492 \end{pmatrix}$$

где три состояния представляют собой «нормальный рост», «мягкую» и «жесткую рецессию».

> James D Hamilton. What's real about the business cycle? Federal Reserve Bank of St. Louis Review, pages 435–452, 2005.

Данную матрицу можно изобразить и графически в виде графа



> T.J. Sargent, J. Stachurski, CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0/

Примем начальное предельное распределение $\psi_0 = (1, 0, 0)$ и сгенерируем цепь длины 50. Сначала делаем все вручную. Обозначим состояния числми 0, 1 и 2, это упростит нам работу.

[1]:
```
import numpy as np
import quantecon as qe
```

Зададим распределение и матрицу P.

[2]:
```
psi = (1.0, 0, 0)
P = [
    [0.971, 0.029, 0.0],
    [0.145, 0.778, 0.077],
    [0, 0.508, 0.492],
]
```

Определим функцию для генерации цепи Маркова. На вход она принимает матрицу вероятностей, начальное предельное распределение и размер генерируемой выборки.

[3]:
```
def mc_sample_path(P, psi_0=None, sample_size=1_000):
    # Преобразовываем P в numpy.array, если она еще таковой не является
    P = np.asarray(P)
    n, _ = P.shape

    # Создадим пустой массив для результатов
    X = np.empty(sample_size, dtype=int)

    # Сгенерируем X_0, либо примем его равным 0, если psi_0 не дано
    if psi_0 is not None:
        X_0 = np.random.choice(range(n), p=psi_0)
    else:
        X_0 = 0

    # Генерируем все X_i
    X[0] = X_0
    for t in range(sample_size - 1):
        X[t + 1] = np.random.choice(range(n), p=P[X[t]])

    return X
```

Посмотрим на результат генерирования цепи длиной 50.

[4]:
```
mc_sample_path(P, psi_0=psi, sample_size=50)
```

[4]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0,
       0, 0, 0, 0, 0, 0])
```

Сгенерируем цепь длиной 1000000 и посмотрим на частоту появления $X_t = 0$.

```
[5]: X = mc_sample_path(P, psi_0=psi, sample_size=1_000_000)
     np.mean(X == 0)
```

```
[5]: np.float64(0.813172)
```

Поменяем начальное распределение ψ_0 и сделаем то же самое.

```
[6]: X = mc_sample_path(P, psi_0=(0.6, 0.3, 0.1),␣
       ↪sample_size=1_000_000)
     np.mean(X == 0)
```

```
[6]: np.float64(0.811553)
```

```
[7]: X = mc_sample_path(P, psi_0=(0.2, 0.6, 0.2),␣
       ↪sample_size=1_000_000)
     np.mean(X == 0)
```

```
[7]: np.float64(0.814147)
```

Вне зависимости от начального распределения ψ_0 доля первого состояния примерно одинакова. Далее покажем, почему. Теперь воспользуемся классом `MarkovChain` из `quantecon`.

```
[8]: mc = qe.MarkovChain(P)
     X = mc.simulate(ts_length=1_000_000)
     np.mean(X == 0)
```

```
[8]: np.float64(0.81325)
```

Этот класс позволяет задавать значения состояний (то есть давать им значения, более конкретные, чем $0, 1, \ldots$)[1] и начальное состояние.

```
[9]: mc = qe.MarkovChain(P, state_values=("normal", "mild",␣
       ↪"severe"))
     mc.simulate(ts_length=15, init="mild")
```

```
[9]: array(['mild', 'normal', 'normal', 'normal', 'mild',␣
       ↪'normal', 'normal',
            'normal', 'normal', 'normal', 'normal', 'normal',␣
       ↪'normal',
            'normal', 'normal'], dtype='<U6')
```

```
[10]: X = mc.simulate(ts_length=1_000_000)
      np.mean(X == "normal")
```

```
[10]: np.float64(0.815592)
```

[1] Строго говоря, это же можно сделать и в нашей самописной функции.

1.2. Предельные распределения

Предположим, что для цепи $\{X_t\}$ мы знаем предельное распределение ψ_t значения X_t. Этого вполне достаточно для определения предельного распределения ψ_{t+1}.

По закону полной вероятности можем записать

$$\Pr\{X_{t+1} = y\} = \sum_{x \in S} \Pr\{X_{t+1} = y \mid X_t = x\} \Pr\{X_t = x\}$$

или в принятых ранее обозначениях

$$\psi_{t+1}(y) = \sum_{x \in S} P(x, y)\psi_t(x)$$

Такое уравнение имеем для каждого $y \in S$.

Объединив все эти уравнения вместе, перепишем все в матрично-векторном виде

$$\psi_{t+1} = \psi_t P$$

где ψ_t, ψ_{t+1} — вектор-строки.

Нетрудно показать, что

$$\psi_{t+m} = \psi_t P^m$$

Также понятно, что элемент p_{ij}^m матрицы P^m равен вероятности перейти из состояния i в состояние j за m шагов. Чтобы это показать наглядно, примем

$$\psi_t = \begin{cases} 1 & \text{для } i \\ 0 & \text{иначе} \end{cases}$$

В приведенном выше примере с безработицей, вероятность оказаться в рецессии через 6 периодов равна

$$\psi P^6 \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

Умножением на столбец мы суммируем вероятности второго и третьего состояний.

1.3. Стационарные распределения

Рассмотрим простой пример

$$P = \begin{pmatrix} 0.4 & 0.6 \\ 0.2 & 0.8 \end{pmatrix}$$

$$\psi = (0.25, 0.75)$$

Нетрудно проверить, что $\psi P = \psi$!

Распределение ψ называется **стационарным** или **инвариантным**.

Каждая стохастическая матрица имеет по крайней мере одно стационарное распределение. Единичная матрица, которая по определению стохастическая, имеет бесконечно много стационарных распределений, так как каждое из них будет таковым.

Если матрица P апериодична и несократима, то у нее есть единственное стационарное распределение.

- **Апериодичность**

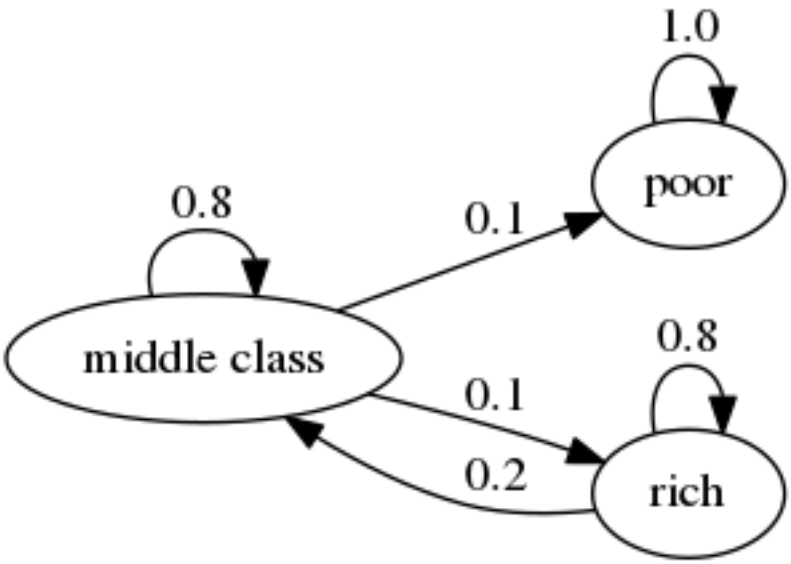
  Если вспомнить пример с безработицей, то мы можем увидеть, что мы можем перемещаться между состояниями по замкнутым траекториям. То есть с некоторой вероятностью мы вернемся в исходное состояние через некоторое количество шагов. Мы можем определить понятие **периода** состояния как наибольший общий делитель набора чисел

  $$D(x) = \{j \geqslant 0 \mid P^j(x, x) > 0\}$$

  $D(x)$ содержит **все** j такие, что из точки x можно вернуться в нее же за j шагов. Если период **всех** состояний равен 1, матрица называется **апериодичной**. Иными словами, мы не должны иметь **детерминированного** характера хождения по циклам!

- **Несократимость (неприводимость)**

  Вернувшись к примеру с безработицей, из каждого состояния мы можем попасть в любое состояние. А вот в этом примере — нет!



  Бедные остаются бедными!

  Если мы можем попасть из любого состояния в любое, или если $\forall x,\ y \in S$ $\exists i,\ j > 0$ такие, что

  $$P^i(x, y) > 0$$
  $$P^j(y, x) > 0$$

  То есть из x в y можно с **ненулевой** вероятностью прийти за i шагов, а обратно — за j.

- **Эргодичность**

  Несократимость влечет за собой следующее следствие

  $$\frac{1}{m}\sum_{t=1}^{m} \mathbf{1}(X_t = x) \xrightarrow{p} \psi^*(x)$$
  $$m \to \infty$$

  то есть доля времени, когда цепь проводит в каком-либо состоянии, стремится к стационарной вероятности.

Объект класса `MarkovChain` включает в себя атрибуты, позволяющие определить апериодичность и неприводимость матрицы P.

[11]:
```
mc = qe.MarkovChain(P)
mc.is_aperiodic, mc.is_irreducible
```

[11]:
```
(True, True)
```

Выведем стационарные распределения для примера с безработицей. Получаемый массив двухмерный, так как стационарных распределений может быть не одно.

[12]:
```
mc.stationary_distributions
```

[12]:
```
array([[0.8128 , 0.16256, 0.02464]])
```

Обратите внимание, что первая вероятность подозрительно похожа на доли первого состояния, которые мы получали выше. Это не случайность, а следствие эргодичности марковской цепи. Воспользуемся этим для написания самописной функции для поиска ψ^*.

[13]:
```
def iter_psi(P, psi_0, tol=1e-6):
    P = np.array(P)
    psi = psi_0
    while True:
        psi_next = psi @ P
        if np.isclose(psi, psi_next, atol=tol).all():
            return psi
        psi = psi_next
```

[14]:
```
iter_psi(P, psi)
```

[14]:
```
array([0.81281945, 0.16254398, 0.02463657])
```

1.4. Математические ожидания

Часто перед нами стоит задача нахождения математических ожиданий для некоторой функции от элементов марковской цепи

$$\mathrm{E}\left[h(X_t)\right]$$

и

$$\mathrm{E}\left[h(X_{t+k}) \mid X_t = x\right]$$

Будем рассматривать $h(\cdot)$ как вектор

$$h = \begin{pmatrix} h(x_1) \\ h(x_2) \\ \vdots \\ h(x_n) \end{pmatrix}$$

Безусловное математическое ожидание определяется просто

$$\mathrm{E}\left[h(X_t)\right] = \sum_{x \in S} (\psi_0 P^t)(x)\, h(x) = \psi_0 P^t h$$

Для рассчета условного математического ожидания $E\left[h(X_{t+k}) \mid X_t = x\right]$ нужно провести аналогичное суммирование по распределению $P^k(x, \cdot)$

$$\mathrm{E}\left[h(X_{t+k}) \mid X_t = x\right] = \sum_{y \in S} P^k(x, y)h(y) = (P^k h)(x) = P^k(x)h$$

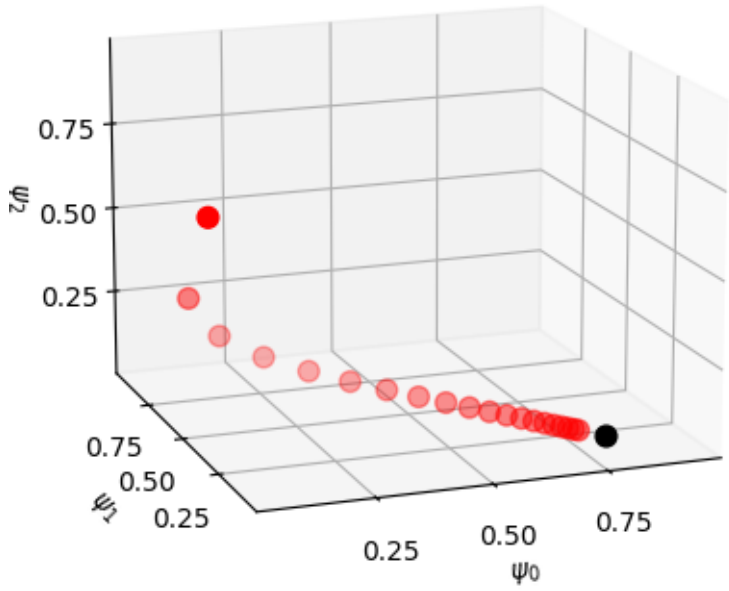
1.5. Сходимость

Рассмотрим (опять) пример с безработицей.

[15]:
```
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (11, 5)
```

Построим график, на котором покажем траекторию предельных распределений.

[16]:
```
P = np.array(P)
psi = (0.0, 0.3, 0.7)

fig = plt.figure()
ax = fig.add_subplot(projection="3d")

ax.set(
    xlim=(0, 1),
    ylim=(0, 1),
    zlim=(0, 1),
    xticks=(0.25, 0.5, 0.75),
    yticks=(0.25, 0.5, 0.75),
    zticks=(0.25, 0.5, 0.75),
)
ax.set_xlabel(r"$\psi_0$")
ax.set_ylabel(r"$\psi_1$")
ax.set_zlabel(r"$\psi_2$")

x_vals, y_vals, z_vals = [], [], []
for t in range(20):
    x_vals.append(psi[0])
    y_vals.append(psi[1])
    z_vals.append(psi[2])
    psi = psi @ P

ax.scatter(x_vals, y_vals, z_vals, c="red", s=60)

mc = qe.MarkovChain(P)
psi_star = mc.stationary_distributions[0]

ax.scatter(psi_star[0], psi_star[1], psi_star[2],
  ↪c="black", s=60)

ax.view_init(15, 250)
plt.show()
```



2. Эксперимент

Рассмотрим модель авторегрессии первого порядка

$$y_{t+1} = \rho y_t + u_{t+1}$$

Из курса эконометрики вам известно, что стационарная дисперсия y равна

$$\sigma_y^2 = \frac{\sigma_u^2}{1 - \rho^2}$$

При помощи метода Тушена данную модель можно аппроксимировать в виде Марковской цепи.

> George Tauchen. Finite state markov-chain approximations to univariate and vector autoregressions. Economics Letters, 20(2):177–181, 1986.

Выберем

- n — число состояний;
- m — ширину пространства состояний.

Создадим пространство состояний такое, что

- $x_0 = -m\sigma_y$
- $x_{n-1} = m\sigma_y$
- $x_{i+1} = x_i + s$, где $s = \frac{x_{n-1} - x_0}{n-1}$

Обозначим через $F(x)$ кумулятивную функцию распределения $N(0, 1)$. Зададим вероятности $P(x_i, x_j)$ следующим образом:

- Если мы переходим в **минимальное** состояние, то

$$P(x_i, x_0) = P\left(\frac{\rho x_i + u_t}{\sigma_u} \leqslant \frac{x_0 + s/2}{\sigma_u}\right) = F(x_0 - \rho x_i + s/2)$$

- Если мы переходим в **максимальное** состояние, то

$$P(x_i, x_{n-1}) = P\left(\frac{x_{n-1} - s/2}{\sigma_u} \leqslant \frac{\rho x_i + u_t}{\sigma_u}\right) = 1 - F(x_{n-1} - \rho x_i - s/2)$$

- Иначе

$$P(x_i, x_j) = P\left(\frac{x_j - s/2}{\sigma_u} \leqslant \frac{\rho x_i + u_t}{\sigma_u} \leqslant \frac{x_j + s/2}{\sigma_u}\right) = F(x_j - \rho x_i + s/2) - F(x_j - \rho x_i - s/2)$$

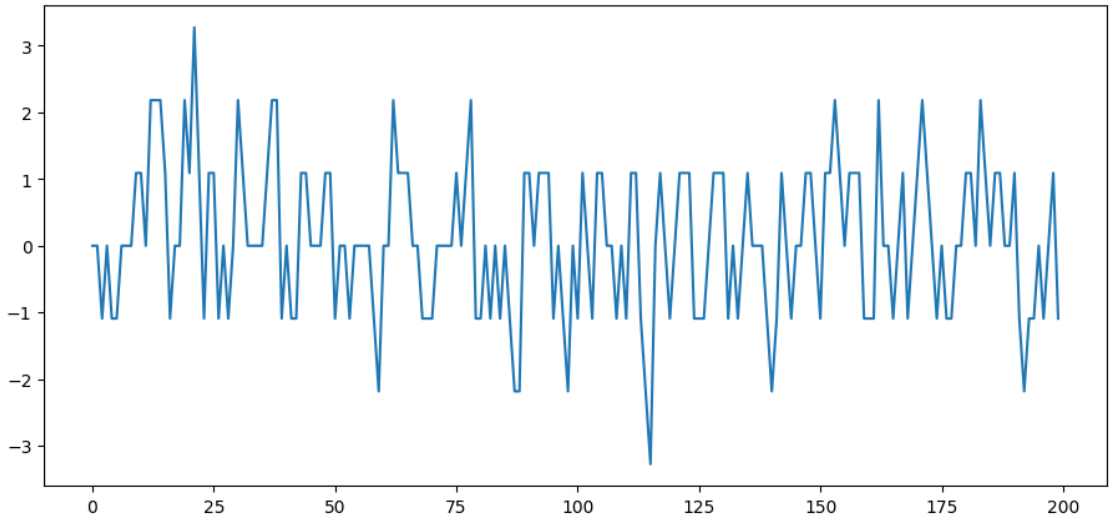
[17]:
```
from scipy.stats import norm
```

Напишем функцию для генерирования дискретной аппроксимацмм процесса $AR(1)$. Будем возвращать объект `MarkovChain`.

[18]:
```
def tauchen(rho, sigma_u=1, m=3, n=7):
    sigma_y = sigma_u / np.sqrt(1 - rho**2)

    x0 = -m * sigma_y
    xN = -x0

    x = np.linspace(x0, xN, n)
    s = (xN - x0) / (n - 1)

    F = lambda x: norm.cdf(x / sigma_u)

    P = np.empty((n, n))

    for i in range(n):
        for j in range(n):
            if j == 0:
                P[i, j] = F(x[0] - rho * x[i] + s / 2)
            elif j == n - 1:
                P[i, j] = 1 - F(x[n - 1] - rho * x[i] - s / 2)
            else:
                P[i, j] = F(x[j] - rho * x[i] + s / 2) -␣
↪F(x[j] - rho * x[i] - s / 2)

    return qe.MarkovChain(P, state_values=x)
```

[19]:
```
armc = tauchen(0.4, sigma_u=1)
x = armc.simulate(200, init=armc.state_values[armc.n // 2])
plt.plot(x)
```

[19]:
```
[<matplotlib.lines.Line2D at 0x1c120fc25d0>]
```



Увеличим число состояний до 100.

[20]:
```
armc = tauchen(0.4, sigma_u=1, n=100)
x = armc.simulate(200, init=armc.state_values[armc.n // 2])
plt.plot(x)
```

[20]:
```
[<matplotlib.lines.Line2D at 0x1c120b6b890>]
```