Национальный исследовательский университет ИТМО
zФакультет информационных технологий и программирования
Прикладная математика и информатика

Марковские процессы

Отчёт по лабораторной работе №3

Работу выполнили:
Гуревич Михаил
Трохан Александр
Соловьёв Роман

Преподаватель:
Москаленко Мария Александровна

Санкт-Петербург 2023

Лабораторная работа №3

Выполнили: Гуревич Михаил, Трохан Александр и Соловьёв Роман, M33001

✅ Полный код лабораторной работы с комментариями можно найти на Github, а отчёт в **Notion**.

Цель работы

Моделирование дискретной эргодической Марковкой цепи.

Задачи

- Придумать эргодическую Марковскую цепь, записать её матрицу, начальный вектор состояний, точность и количество шагов.
- Промоделировать цепь пошагово с двумя начальными векторами состояний, получить графики изменение среднеквадратичного отклонения и два конечных вектора.
- Решить задачу аналитически и получить вектор.
- Определить, схожи ли вектора из пошагового и аналитического метода решения между собой.

Ход работы

Для начала придумаем некоторый Марковский процесс, который будет описываться следующей матрицей размера 8 на 8:

$$P = \begin{pmatrix} 0.2 & 0.3 & 0.1 & 0 & 0 & 0.2 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.1 & 0.2 & 0.1 & 0.1 & 0.1 & 0 \\ 0.1 & 0.1 & 0.4 & 0.1 & 0.1 & 0.1 & 0 & 0.1 \\ 0 & 0.2 & 0.1 & 0.2 & 0.1 & 0.1 & 0.2 & 0.1 \\ 0.1 & 0 & 0.1 & 0.2 & 0.3 & 0.2 & 0.1 & 0 \\ 0.1 & 0.1 & 0.2 & 0.1 & 0.2 & 0.2 & 0.1 & 0 \\ 0.2 & 0.1 & 0.1 & 0.2 & 0.1 & 0.2 & 0.1 & 0 \\ 0.1 & 0.1 & 0.2 & 0.1 & 0.1 & 0.2 & 0.1 & 0.1 \end{pmatrix}$$

Также выберем два начальных вектора вероятностей s_1 и s_2 для пошагового метода и точность ε:

$$s_1 = (1\ 0\ 0\ 0\ 0\ 0\ 0\ 0)$$
$$s_2 = \left(\frac{1}{8}\ \frac{1}{8}\ \frac{1}{8}\ \frac{1}{8}\ \frac{1}{8}\ \frac{1}{8}\ \frac{1}{8}\ \frac{1}{8}\right)$$
$$\varepsilon = 5 \cdot 10^{-6}$$

Реализация методов

Методы инициализации кастомного класса для игры и ввода данных из json’а

Описание метода: реализуем класс `MarkovChain`, который будет хранить в себе матрицу процесса и базовые методы для вычисления векторов вероятностей. Инициализировать класс можно из json-файла, где задана матрица и точность вычисления.

Реализация метода:

```
def __init__(self, matrix: np.ndarray, eps: decimal.Decimal, max_step: int=MAX_STEP):
    self.matrix = matrix.copy()
    self.eps = eps
    self.max_step = max_step

@staticmethod
def from_json(path: str):
    with open(path) as f:
        data = json.load(f)
        matrix = np.array(data["transition_matrix"])
```

```
        eps = round(decimal.Decimal(data["epsilon"]), 10)
    return Markov_chain(matrix, eps)
```

Метод проверки цепи на эргодичность

Описание метода: данный метод проверяет, является ли цепь, которую описывает класс, эргодической. Достаточным условием эргодичности является наличие в графе цепи одной компоненты связности, что и проверяется в методе (реализовано в отдельном классе `Graph`). Также граф не цикличен, так как из любого состояния можно вернуться с некоторой вероятностью (по условию). Помимо этого присутствует проверка на то, является ли матрица квадратной и что сумма элементов строк равна единице.

Реализация метода:

```
def _check_ergodicity(self):
    """Check if matrix is ergodic (indecomposable + aperiodic state)."""

    g = Graph(self.matrix)
    if g.find_striongly_connected_components_count() != 1:
        raise ValueError("The Markov chain is not ergodicity.")

    for i in range(len(self.matrix)):
        if len(self.matrix) != len(self.matrix[i]):
            raise ValueError("Incorrect shape of matrix.")

        sum_ = 0
        for j in range(len(self.matrix[i])):
            if self.matrix[i][j] < 0:
                raise ValueError("The probability cannot be less than 0.")

            if i == j and self.matrix[i][j] == 0:
                raise ValueError("The probability of staying must be greater than 0.")

            sum_ += decimal.Decimal(self.matrix[i][j])

        if sum_ != 1:
            raise ValueError("The sum of probabilities in each row must be equal to 1.")
```

Численное нахождение распределения по состояниям

Описание метода: метод `_numeric_model` рекурсвино вычисляет произвдение заданного вектора вероятностей на матрицу, и при достижении необходимой

точности возвращает полученный вектор, число итераций, а также список среднеквадратичных отклонений. Число итераций также ограничено заранее заданной константой `self.max_step`.

Затем метод `get_numeric_distribution` по этим данным строит график среднеквадратичного отклонения и возвращает результат.

Реализация метода:

```
def _numeric_model(self, s: np.ndarray, step: int, shape: list=[]):
    """Find result distribution vector and MSE using iteration method."""

    if step > self.max_step:
        return (s, step - 1, shape)

    s_next: np.ndarray = s @ self.matrix
    shape.append((np.square(s - s_next)).mean())

    if la.norm(np.abs(s_next - s)) < self.eps:
        return (s_next, step, shape)
    else:
        return self._numeric_model(s_next, step + 1, shape)

def get_numeric_distribution(self, s: np.ndarray):
    """Handles iterations result and get a graph of MSE changing from it."""

    if np.sum(s) != 1:
        raise ValueError("The sum of probabilities in distribution must be equal to 1.")

    s_res, n, shape = self._numeric_model(s, 1, [])

    fig = plt.figure()
    plt.plot(range(1, n + 1), shape)
    return (s_res, n, fig)
```

Аналитическое нахождение распределения по состояниям

Описание метода: для аналитического нахождения распределения заданного вектором s требуется решить систему следующего вида:

$$\begin{cases} s_1 = s_1p_{11} + s_2p_{21} + ... + s_np_{n1} \\ s_2 = s_1p_{21} + s_2p_{22} + ... + s_np_{n2} \\ ... \\ s_n = s_1p_{1n} + s_2p_{2n} + ... + s_np_{nn} \end{cases}$$

Так как данная система имеет n переменных и n уравнений, для неё существует однозначное решение. В данном случае нетрудно заметить, что это будет решение $s = (0\ 0\ \ldots\ 0)$. Поэтому заменим одно из уравнений на сумму s_i, которая должна равняться 1, так как это вероятности, а также перенесём свободные члены в правую часть:

$$\begin{cases} s_1 + s_2 + ... + s_n = 1 \\ s_1 p_{21} + s_2(p_{22} - 1) + ... + s_n p_{n2} = 0 \\ ... \\ s_1 p_{1n} + s_2 p_{2n} + ... + s_n(p_{nn} - 1) = 0 \end{cases}$$

Такую систему можно решить методом Гаусса, чем и занимается реализованный нами метод.

Реализация метода:

```
def _make_identity(self, matrix: np.ndarray):
    """Creates identity matrix after Gauss."""

    for nrow in range(len(matrix) - 1, 0, -1):
        row = matrix[nrow]
        for upper_row in matrix[:nrow]:
            factor = upper_row[nrow]
            upper_row -= factor * row

    return matrix

def _gauss(self, matrix: np.ndarray):
    """Gauss method."""

    for nrow in range(len(matrix)):
        pivot = nrow + np.argmax(abs(matrix[nrow:, nrow]))
        if pivot != nrow:
            matrix[[nrow, pivot]] = matrix[[pivot, nrow]]
        row = matrix[nrow]
        divider = row[nrow]
        if abs(divider) < 1e-10:
            raise ValueError("Matrix is singular.")

        row /= divider

        for lower_row in matrix[nrow+1:]:
            factor = lower_row[nrow]
            lower_row -= factor * row
```

```
    return self._make_identity(matrix)

def get_analytical_distribution(self):
    """Find result distribution vector using analytical method (with Gauss)."""

    M = np.copy(self.matrix.T) - np.eye(len(self.matrix))
    M[0] = np.ones(len(M[0]))
    b = np.zeros(len(M))
    b[0] = 1
    M = np.insert(M, len(M), b, axis=1)
    print(M)

    return (self._gauss(M))[:, len(M)]
```

Результат работы

Код был запущен с входными данными, которые были приведены выше, и были получены следующие результаты:

Ветор стационарного распределения полученный пошагово из состояния s_1: $s = (0.1219\ 0.1392\ 0.1718\ 0.1381\ 0.1291\ 0.1551\ 0.0966\ 0.0479)$

Ветор стационарного распределения полученный пошагово из состояния s_2: $s = (0.1219\ 0.1392\ 0.1718\ 0.1381\ 0.1291\ 0.1550\ 0.0966\ 0.0479)$

График среднеквадратичной ошибки для вектора s_1:

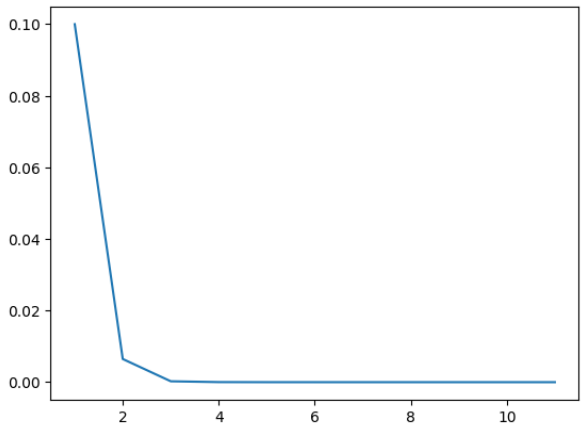
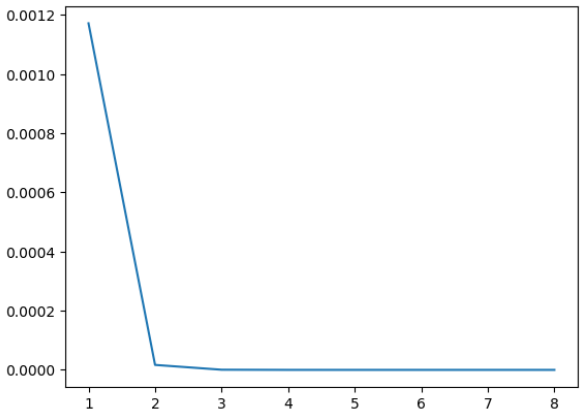


График среднеквадратичной ошибки для вектора s_2:



Ветор стационарного распределения полученный аналитически: $s = (0.1219\ 0.1392\ 0.1718\ 0.1381\ 0.1291\ 0.1550\ 0.0966\ 0.0479)$

Вектора стационарных распределений совпадают с высокой точностью.

Вывод

В ходе данной лабораторной работы была продемонстрирована реализация и работа методов нахождения стационарного распределения в марковской цепи. Полученные результаты подтверждают корректность реализации, и показывают, что оба метода дают близкие результаты.