Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

По лабораторной работе №2

По курсу: «Моделирование»

Тема: «Марковские процессы»

Студент: Ле Ни Куанг

Группа: ИУ7И-76Б

Преподаватель: Рудаков И. В.

Москва

2021

1 Задание

Написать программу позволяет определить время стабилизации сложной системы для каждого состояния. Количество состояний не более 10. На вход подается граф-матрица, на пересечении строк и столбцов которой находится интенсивность перехода.

2 Теоритическая часть

Случайный процесс, протекающий в сложной системе S, называется марковским, если для каждого момента времени t_0 вероятность любого состояния системы в будущем зависит только от состояния системы в настоящем (т.е. не зависит от того, когда и каким образом система перешла в это состояние).

В системе n состояний $\{S_1, \ldots, S_n\}$.

Матрица интенсивностей:

$$\Lambda = \begin{pmatrix} \lambda_{11} & \lambda_{12} & \ldots & \lambda_{1n} \\ \lambda_{21} & \lambda_{22} & \ldots & \lambda_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{n1} & \lambda_{n2} & \ldots & \lambda_{nn} \end{pmatrix}$$

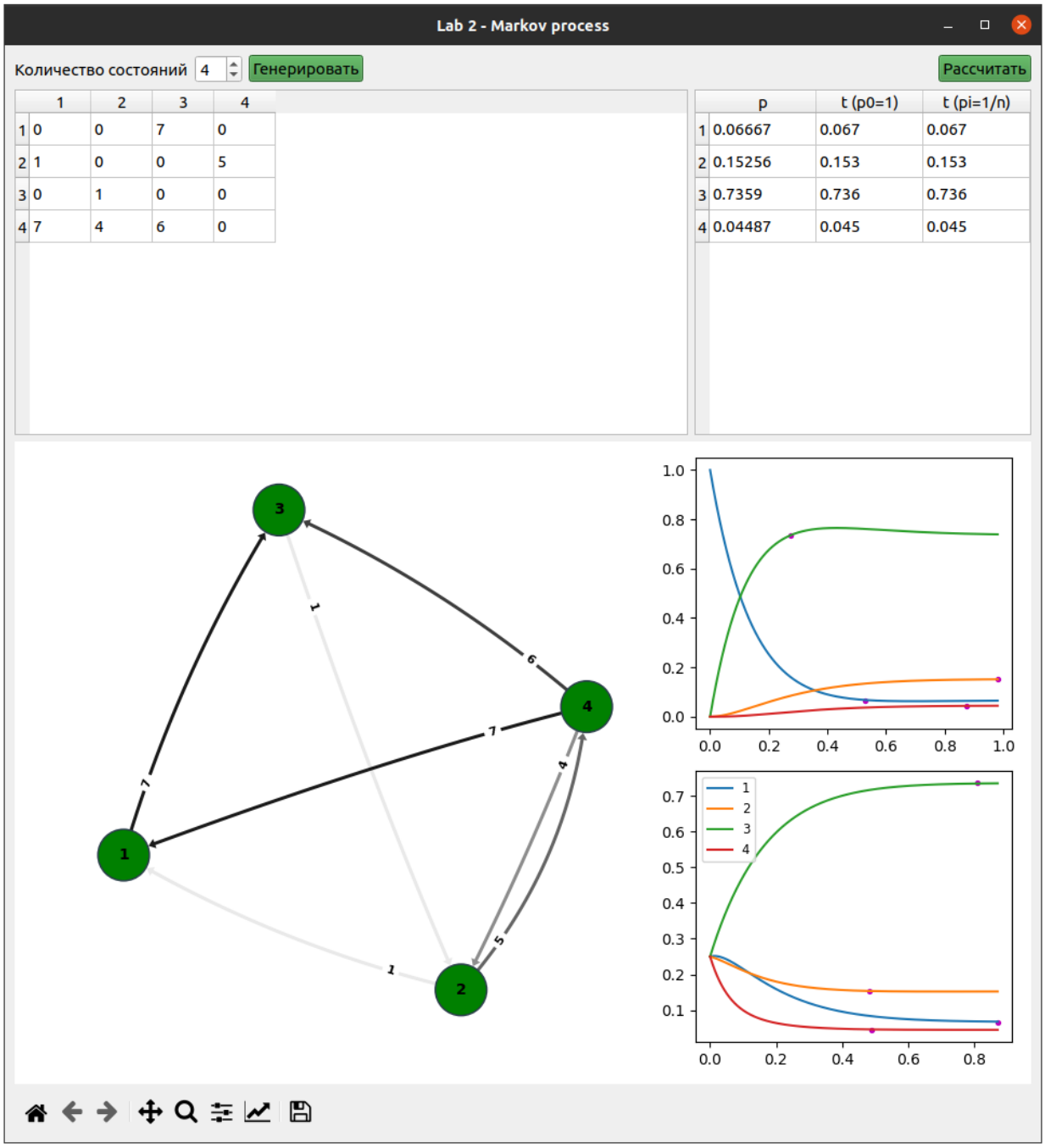
Уравнение Колмогорова:

$$\frac{\mathrm{d}p_i(t)}{\mathrm{d}t} = \sum_{j=1}^{n} \lambda_{ji} p_j(t) - p_i(t) \sum_{j=1}^{n} \lambda_{ij}.$$

Уравнение нормировки:

$$\sum_{i=1}^{n} p_i(t) = 1$$

3 Результаты

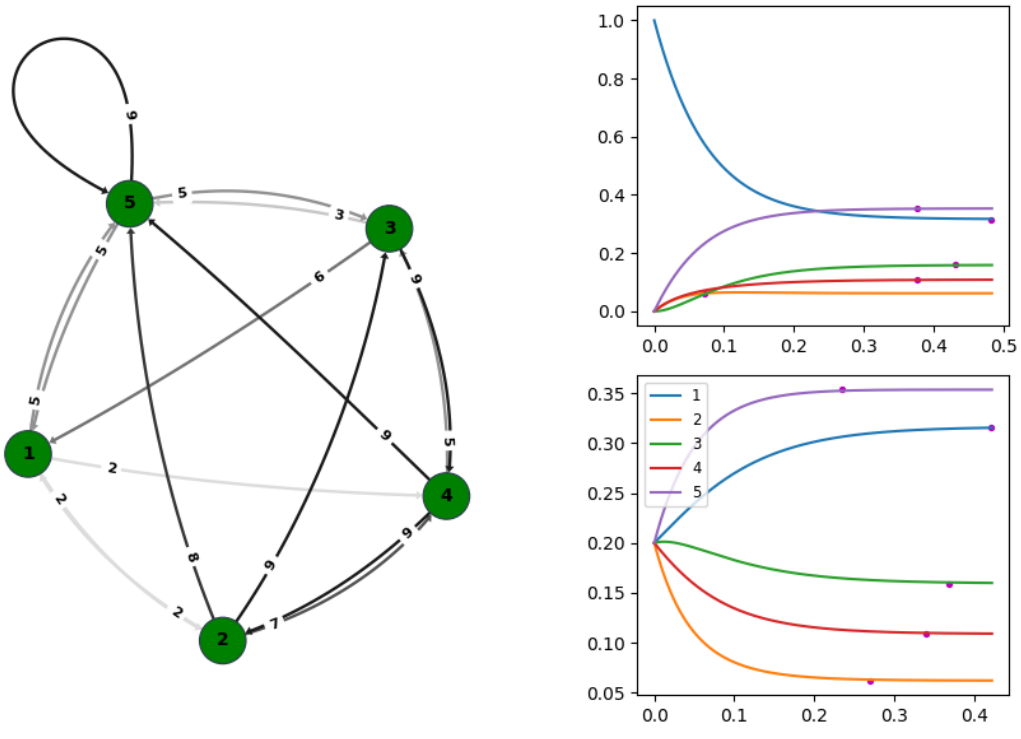


Lab 2 - Markov process

Количество состояний 5 Генерировать Рассчитать

	1	2	3	4	5
1	0	2	0	2	5
2	2	0	9	7	8
3	6	0	0	9	3
4	0	9	5	0	9
5	5	0	5	0	9

	p	t (p0=1)	t (pi=1/n)
1	0.31641	0.316	0.316
2	0.06198	0.062	0.062
3	0.15939	0.159	0.159
4	0.10875	0.109	0.109
5	0.35348	0.353	0.353

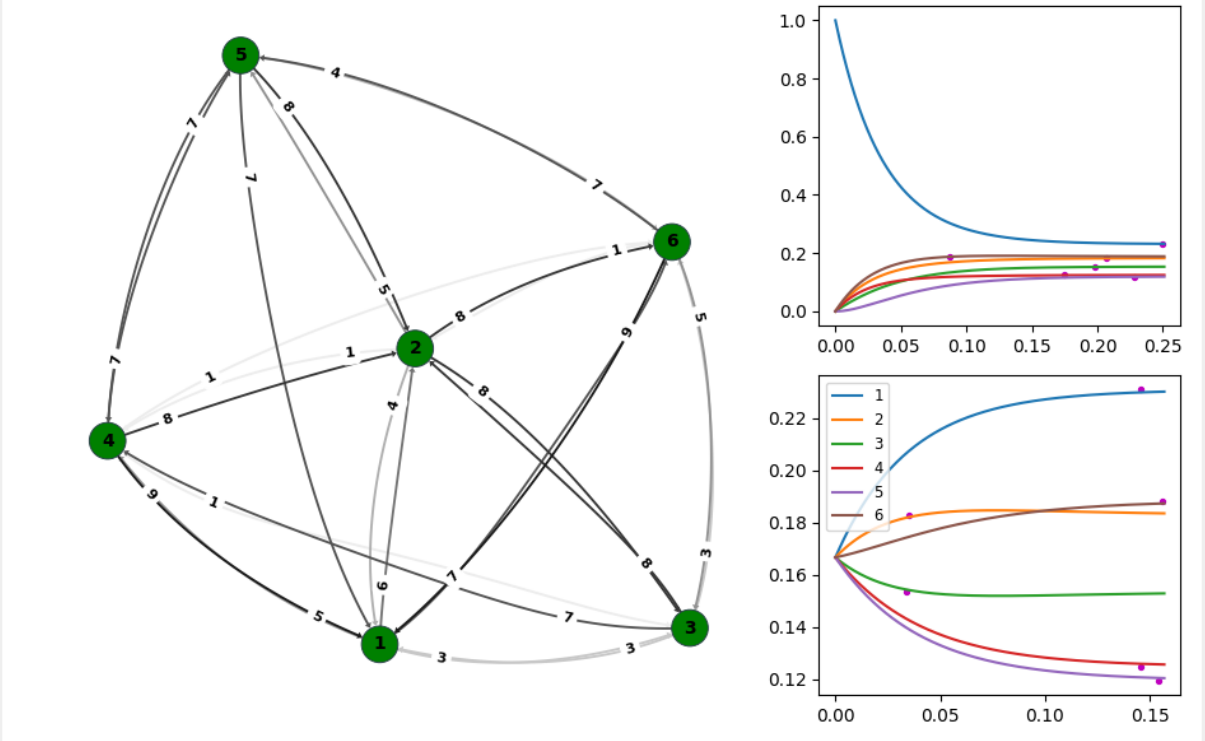


Lab 2 - Markov process

Количество состояний 6 Генерировать Рассчитать

	1	2	3	4	5	6
1	0	6	3	5	0	7
2	4	0	8	1	5	8
3	3	8	0	7	0	3
4	9	8	1	0	7	1
5	7	8	0	7	0	4
6	9	1	5	0	7	0

	p	t (p0=1)	t (pi=1/n)
1	0.23085	0.231	0.231
2	0.18294	0.183	0.183
3	0.15346	0.153	0.153
4	0.12492	0.125	0.125
5	0.11951	0.12	0.12
6	0.18831	0.188	0.188

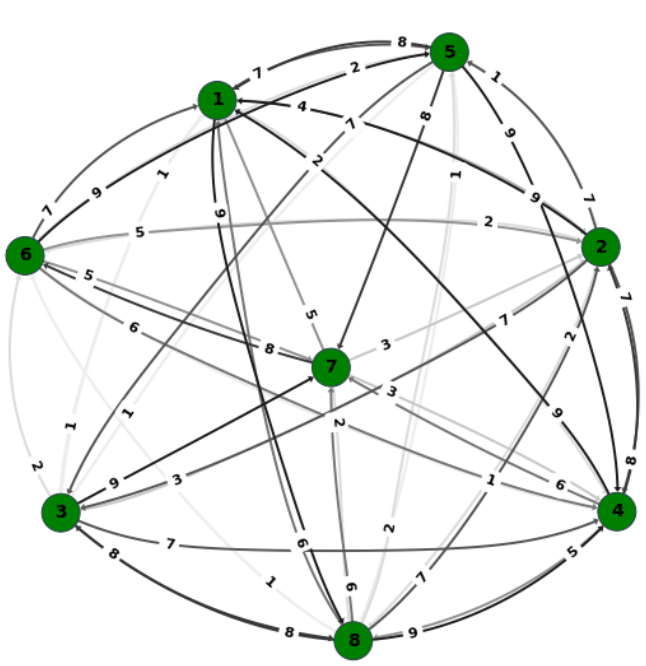


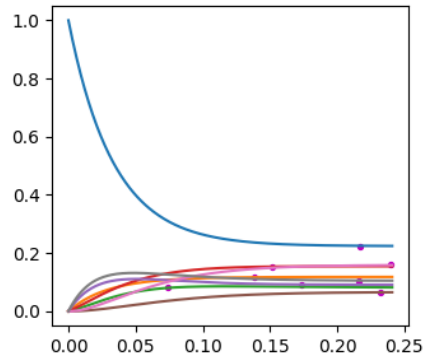
Lab 2 - Markov process

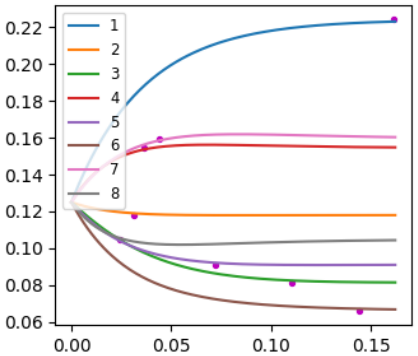
Количество состояний 8 Генерировать Рассчитать

	1	2	3	4	5	6	7	8
1	0	4	1	2	7	0	0	9
2	9	0	7	7	7	2	0	2
3	1	3	0	7	1	2	9	8
4	9	8	0	0	0	1	6	5
5	8	1	7	9	0	2	8	1
6	7	5	0	6	9	0	5	0
7	5	3	0	3	0	8	0	2
8	6	7	8	9	2	1	6	0

	p	t (p0=1)	t (pi=1/n)
1	0.22411	0.224	0.224
2	0.11802	0.118	0.118
3	0.08152	0.082	0.082
4	0.15464	0.155	0.155
5	0.09116	0.091	0.091
6	0.06616	0.066	0.066
7	0.15955	0.16	0.16
8	0.10484	0.105	0.105







4 Листинг кода

Листинг 1 – Программная реализация определения времени пребывания сложной системы в каждом из состояний

```
import numpy as np
from random import random, randrange
from matplotlib import pyplot as plt

EPS = 1e-3
DT = 1e-3


class Markov:
    MaxSize = 10

    def __init__(self):
        self.graph = np.array([])
        self.result = np.array([])

    def gen_state_graph(self, size):
        size = size if size < self.MaxSize else self.MaxSize
        self.result = np.zeros((size, 3))
        self.graph = np.random.randint(10, size=(size, size),
           dtype=np.int)
        for i in range(size):
            if random() > 0.2:
                self.graph[i, i] = 0
            # randomly reduce number of edges
            self.graph[randrange(size), randrange(size)] = 0

    def solve(self, axs=[]):
        m = self.graph
        n = m.shape[0]

        coeff_kolmogorov = m.copy().T
        for i in range(n):
            coeff_kolmogorov[i, i] -= sum(m[i])

        p_stable = self.solve_p(m, coeff_kolmogorov.copy())
        self.result[:, 0] = p_stable.round(5)

        p_init = np.array([1] + [0] * (n - 1), dtype=float)
        t1, p1 = self.solve_t(coeff_kolmogorov.copy(), p_stable, p_init)
        self.result[:, 1] = p_stable.round(3)

        p_init = np.array([1 / n] * n)
        tn, pn = self.solve_t(coeff_kolmogorov.copy(), p_stable, p_init)
```

```
        self.result[:, 2] = p_stable.round(3)

        if len(axs) == 2:
            label = list(range(1, n + 1))

            n = p1.shape[0]
            t = np.linspace(0, DT * n, n)
            self.plot(t, p1, axs[0], label=label)
            axs[0].scatter(t1, p_stable, s=8, c='m')
            n = pn.shape[0]
            t = np.linspace(0, DT * n, n)
            axs[1].scatter(tn, p_stable, s=8, c='m')
            self.plot(t, pn, axs[1], label=label)

    @staticmethod
    def plot(x, y, axs, *, label=None):
        lines = axs.plot(x, y)
        plt.legend(lines, label, loc='upper left', fontsize='small')

    @staticmethod
    def solve_p(graph, coeff_kolmogorov) -> np.ndarray:
        n = graph.shape[0]
        a = coeff_kolmogorov
        b = np.zeros(n)

        s0 = sum(graph[0])
        b[0] = s0
        for i in range(n):
            a[0, i] += s0
        return np.linalg.solve(a, b)

    @staticmethod
    def solve_t(coeff_kolmogorov, p_stable, p_init):
        n = coeff_kolmogorov.shape[0]
        t_stable = [0] * n
        p_cur = p_init
        p_trace = np.array([p_cur])

        def update_stability(p_c, dp, p_s, t):
            if abs(dp) < EPS and abs(p_c - p_s) < EPS:
                return t
            return 0

        t = 0
        while not all(t_stable):
            t += DT
            dp = (coeff_kolmogorov @ p_cur) * DT
            p_cur += dp
            p_trace = np.append(p_trace, [p_cur], axis=0)
```

```
            for i in range(n):
                if t_stable[i] == 0:
                    t_stable[i] = update_stability(p_cur[i], dp[i],
                        p_stable[i], t)

        return t_stable, p_trace
```