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»

на тему: «Цепи Маркова»

Студент ИУ7-73Б (Группа) _________ (Подпись, дата) Миронов Г. А. (И. О. Фамилия)

Преподаватель _________ (Подпись, дата) Рудаков И. В. (И. О. Фамилия)

2022 г.

1 Задание

Реализовать программу, позволяющую определить время пребывания сложной системы, работающей на базе цепей Маркова, во всех ее состояниях, определить момент достижения вероятностной константы, а также ее значение.

2 Теоретическая часть

2.1 Марковский процесс

Случайный процесс, протекающий в системе S, называется Марковским, если он обладает следующим свойством: для каждого момента времени t_0 вероятность любого состояния системы в будущем (при $t > t_0$) зависит только от ее состояния в настоящем (при $t = t_0$) и не зависит от того, когда и каким образом система пришла в это состояние.

Вероятность того, что в момент t система будет находиться в состоянии S_i, называется вероятностью i-го состояния. Для любого момента t сумма вероятностей всех состояний равна единице.

2.2 Вычисление вероятностной константы

Для решения поставленной задачи, необходимо составить систему уравнений Колмогорова по следующим принципам:

- в левой части каждого из уравнений стоит производная вероятности i-го состояния;
- в правой части — сумма произведений вероятностей всех состояний, из которых идут стрелки в данное состояние, умноженная на интенсивности соответствующих потоков событий, минус суммарная интенсивность всех потоков, выводящих систему из данного состояния, умноженная на вероятность данного состояния.

2.3 Пример

Пусть дана матрица интенсивностей для системы S, имеющей три состояния:

$$\begin{pmatrix} 0 & \lambda_{01} & \lambda_{02} \\ \lambda_{10} & 0 & \lambda_{12} \\ \lambda_{20} & \lambda_{21} & 0 \end{pmatrix} \tag{2.1}$$

Тогда соответствующая система уравнений Колмогорова имеет вид:

$$\begin{cases} p_0' = -(\lambda_{01} + \lambda_{02})p_0 + \lambda_{10}p_1 + \lambda_{20}p_2 \\ p_1' = -(\lambda_{10} + \lambda_{12})p_1 + \lambda_{01}p_0 + \lambda_{21}p_2 \\ p_2' = -(\lambda_{20} + \lambda_{21})p_2 + \lambda_{02}p_0 + \lambda_{12}p_1 \end{cases} \tag{2.2}$$

Для нахождения вероятностных констант, то есть вероятностей в стационарном режиме работы при $t- > \inf$, необходимо приравнять левые части уравнений к нулю.

Таким образом получается система линейных уравнений. Для решения полученной системы необходимо добавить условие нормировки:

$$\sum_i p_i = 1 \tag{2.3}$$

Кроме того, необходимо так же найти время, за которое достигается вероятностная константа.

В общем случае для решения данной задачи необходимо решить систему ОДУ 2.2 в общем виде.

Для решения данной задачи можно воспользоваться численным методом: найдем все значения вероятности как функции времени, с шагом в некотором интервале $[t_0, t_N]$. Когда вычисленная вероятность будет равна найденной ранее вероятностной константе с точностью до заданной погрешности, можно считать что искомое время найденно.

3 Результат работы

3.1 Общий случай

В таблице 3.1 приведена матрица интенсивностей для рассматриваемой системы.

Таблица 3.1 – Описание системы

Из / В	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
S_1	0.83	0.15	0.37	0.89	0.05	0.11	0.27	0.01
S_2	0.39	0.86	0.98	0.84	0.91	0.93	0.94	0.06
S_3	0.77	0.26	0.11	0.13	0.48	0.68	0.75	0.17
S_4	0.12	0.09	0.70	0.95	0.96	0.53	0.76	0.23
S_5	0.79	0.77	0.86	0.59	0.80	0.55	0.71	0.60
S_6	0.64	0.32	0.34	0.04	0.96	0.86	0.02	0.21
S_7	0.24	0.60	0.65	0.26	0.20	0.35	0.93	0.39
S_8	0.61	0.63	0.27	0.94	0.81	0.49	0.12	0.55

На рисурке 3.1 представлен граф, описывающий связи внутри рассматриваемой системы.

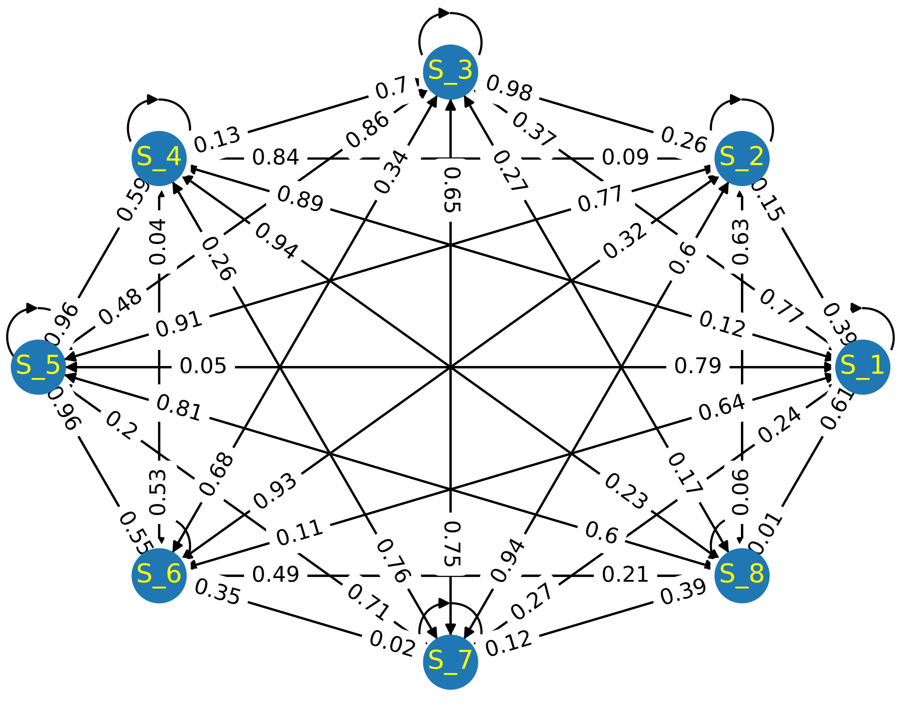

Рисунок 3.1 – Граф, описывающий систему

На рисунке 3.2 представлен график зависимости вероятности каждого из состояний от времени.

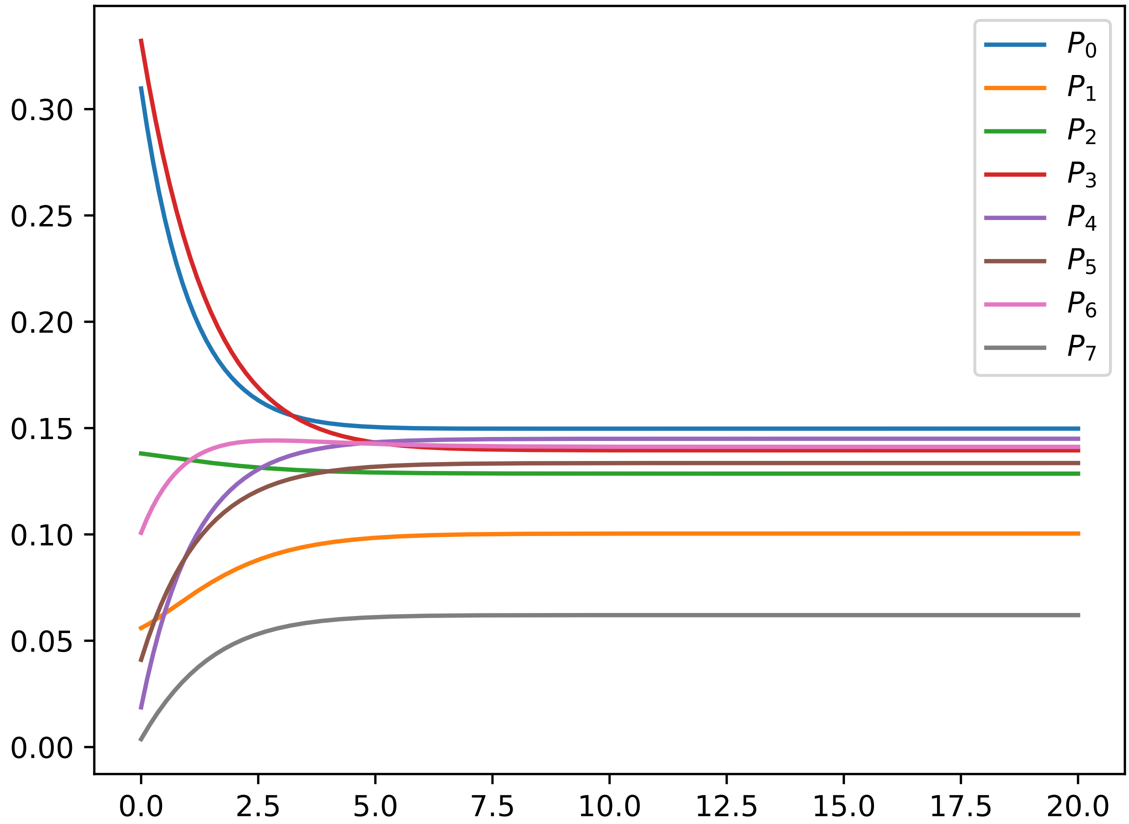

Рисунок 3.2 – Зависимость вероятности от времени

В таблице 3.2 представлены полные результаты работы программы для рассматрвиаемой системы.

Таблица 3.2 – Результаты работы

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
Стабильное состояние	0.150	0.100	0.129	0.140	0.145	0.134	0.141	0.062
Время	14.381	12.971	12.655	12.813	13.447	13.291	14.761	12.157

3.2 Кольцо

В таблице 3.3 приведена матрица интенсивностей для рассматриваемой системы.

Таблица 3.3 – Описание системы

Из / В	S_1	S_2	S_3	S_4
S_1	0.00	0.80	0.00	0.00
S_2	0.00	0.00	0.30	0.00
S_3	0.00	0.00	0.00	0.26
S_4	0.53	0.00	0.00	0.00

На рисурке 3.3 представлен граф, описывающий связи внутри рассматриваемой системы.

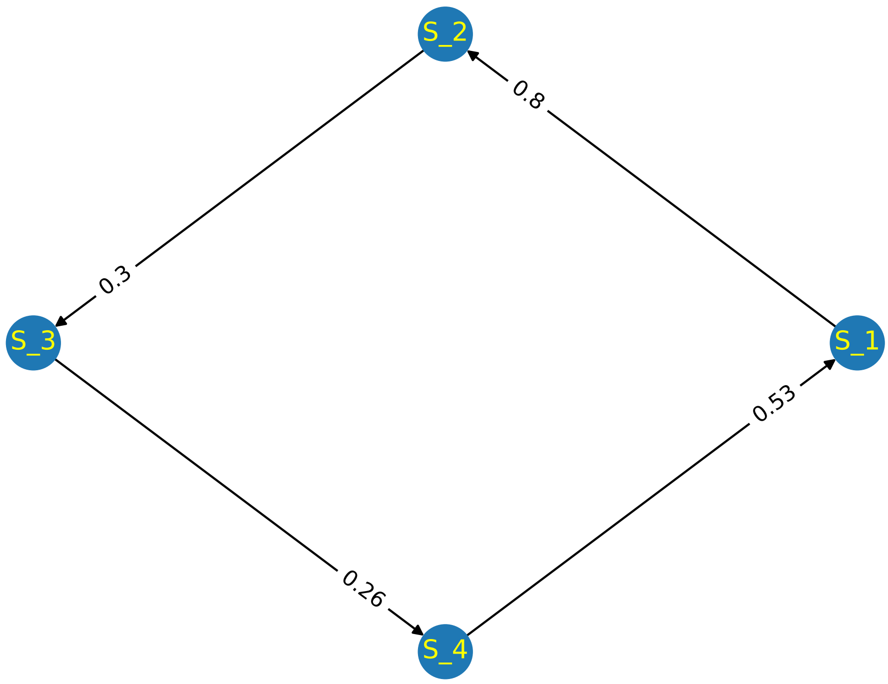

Рисунок 3.3 – Граф, описывающий систему

На рисунке 3.4 представлен график зависимости вероятности каждого из состояний от времени.

В таблице 3.4 представлены полные результаты работы программы для рассматрвиаемой системы.

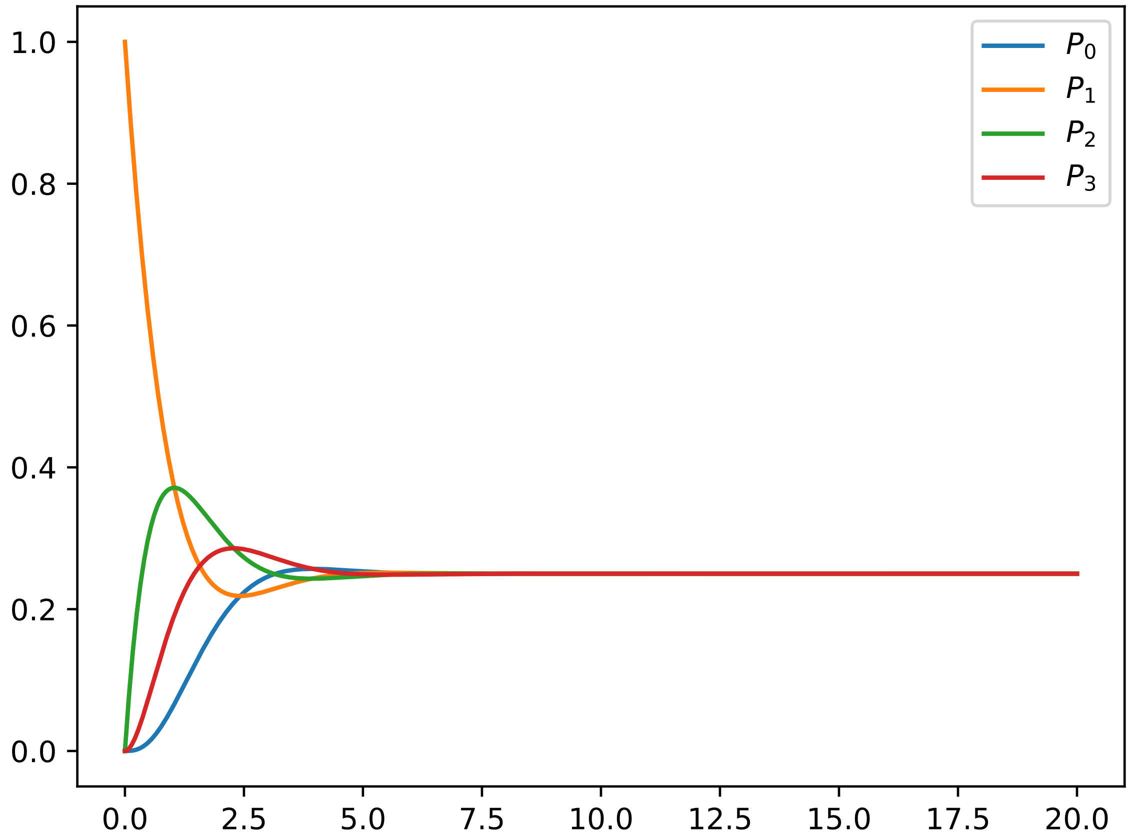

Рисунок 3.4 – Зависимость вероятности от времени

Таблица 3.4 – Результаты работы

	S_1	S_2	S_3	S_4
Стабильное состояние	0.250	0.250	0.250	0.250
Время	12.171	13.003	12.171	13.003

4 Исходный код программы

Листинг 4.1 – Исходный код программы. Часть 1

```
import random
import time

import matplotlib.pyplot as plt
import scipy.integrate as integrate
import numpy as np
import pandas as pd
import networkx as nx

random.seed(time.time())

import ipywidgets as widgets

states_number_input = widgets.BoundedIntText(value=0, min=0, max=100, step=1, layout=widgets.Layout(width="50px"))

randomize_input = widgets.Checkbox(value=False, description='Случайное␣заполнение', indent=False)

display(widgets.Label("Количество␣состояний␣в␣системе"))
display(states_number_input)
display(randomize_input)

states_n = states_number_input.value

matrix_inputs = []
items = []

items += [widgets.Label("Состояния")]
items += [widgets.Label(f"$$\\text{{в␣}}S_{{{i+1}}}$$") for i in range(states_n)]

for i in range(states_n):
    items += [widgets.Label(f"$$\\text{{из␣}}S_{{{i+1}}}$$")]

    inputs = [
        widgets.BoundedFloatText(value= 0 if not randomize_input.value else round(random.random(), 2), min=0, max=1, step=0.01, layout=widgets.Layout(width="50px"))
        for _ in range(states_n)
    ]
    matrix_inputs += [inputs]
    items += inputs
```

Листинг 4.2 – Исходный код программы. Часть 2

```
display(widgets.GridBox(items, layout=widgets.Layout(grid_template_columns=f"repeat({states_n + 1}, 1fr)", overflow="auto")))

def get_matrix():
    return np.array([[float(matrix_inputs[i][j].value) for j in range(states_n)] for i in range(states_n)])

m = get_matrix()

G = nx.from_numpy_matrix(m, create_using=nx.DiGraph)
layout = nx.circular_layout(G)

nx.draw(G, layout, node_size=600)
nx.draw_networkx_labels(G, pos=layout, labels={i: f'S_{i+1}' for i in range(states_n)}, font_color='yellow', font_size=12)
nx.draw_networkx_edge_labels(G, pos=layout, edge_labels=nx.get_edge_attributes(G,'weight'), label_pos=0.2)
plt.show()

t_max_input = widgets.BoundedFloatText(value=0, min=0, max=1000, step=1, description="Tmax")
t_n_input = widgets.BoundedIntText(value=0, min=0, max=10e4, step=1, description="N")

display(widgets.Label("Время моделирования"))
display(t_max_input)
display(t_n_input)

eps = 1e-6
dt = 0.1
tn = t_n_input.value
tmax = t_max_input.value

m = get_matrix()

m = m / m.sum(axis=1).reshape((-1, 1))

def find_stable(matrix):
    b = [0] * (len(matrix) - 1) + [1]

    matrix_to_solve = matrix.copy().transpose()
    matrix_to_solve -= np.diag(matrix.sum(axis=1))
    matrix_to_solve[-1] = np.ones(len(matrix_to_solve))

    return np.linalg.solve(matrix_to_solve, b)
```

Листинг 4.3 – Исходный код программы. Часть 3

```
def solve_ode(matrix, start_probs, tn, tmax, steady_states):
    matrix_to_solve = matrix.copy().transpose()
    matrix_to_solve -= np.diag(matrix.sum(axis=1))

    ts = np.linspace(0, tmax, tn)

    results = integrate.odeint(vectorfields, start_probs, ts,
       args=(matrix_to_solve,), atol=1.0e-8, rtol=1.0e-6).
       transpose()

    steady_ts = []

    for i in range(len(results)):
        row = results[i]
        flag = True
        for j in range(len(row) - 1, -1, -1):
            if abs(steady_states[i] - row[j]) > eps:
                steady_ts.append(ts[j])
                flag = False
                break
        if flag:
            steady_ts.append(0)

    return ts, results, steady_ts

def vectorfields(w, _, matrix_to_solve):
    f = []
    for i in range(len(w)):
        f.append(0)
        for p, lambda_coeff in zip(w, matrix_to_solve[i]):
            f[i] += p * lambda_coeff

    return f

def find_stable_time(matrix, start_probs, tn, tmax):
    ps = find_stable(matrix)
    return solve_ode(matrix, start_probs, tn, tmax, ps)
```

Листинг 4.4 – Исходный код программы. Часть 4

```
stable = find_stable(m).reshape((1, -1))

start_state_probs = m[0]

ts, data, steady = find_stable_time(m, start_state_probs, tn, tmax)

df = pd.DataFrame.from_dict({'Стабильное состояние': find_stable(m), 'Время': steady}, orient='index', columns=[f"$$S_{{{i+1}}}$$" for i in range(states_n)])
df.style.format("{:.3f}")
df.to_latex(float_format="%.3f")

for i in range(len(data)):
    plt.plot(ts, data[i], label=f"$P_{{{i}}}$")
plt.legend()
plt.show()
```