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

к лабораторной работе №2

По курсу: «Моделирование»

Студент ***ИУ7И-76Б*** _____ **Нгуен Ф. С.**
(Группа) (Подпись, дата) (И.О. Фамилия)

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

Москва, 2021 г.

Оглавление

I. Теоретическая часть

Случайный процесс, протекающий в системе S, называется марковским, если он обладает следующим свойством: для каждого момента времени $t0$ вероятность любого состояния системы в будущем (при $t > t0$) зависит только от ее состояния в настоящем (при $t = t0$) и не зависит от того, когда и каким образом система пришла в это состояние. Вероятностью i-го состояния называется вероятность $pi(t)$ того, что в момент t система будет находиться в состоянии Si. Для любого момента t сумма вероятностей всех состояний равна единице.

Уравнения Колмогорова в общем виде:

$$\frac{dp_i(t)}{dt} = \sum_{j-1}^{n} \quad p_j(t)\lambda_{ji} - p_i(t) \sum_{j-1}^{n} \quad \lambda_0 \qquad\qquad i = 1, \ldots, n \tag{1}$$

Имея в распоряжении размеченный граф состояний, можно найти все вероятности состоянийр_i (t)как функции времени. Для этого составляются и решаются так называемые уравнения Колмогорова особого вида дифференциальные уравнения, в которых неизвестными функциями являются вероятности состояний.

Правило:

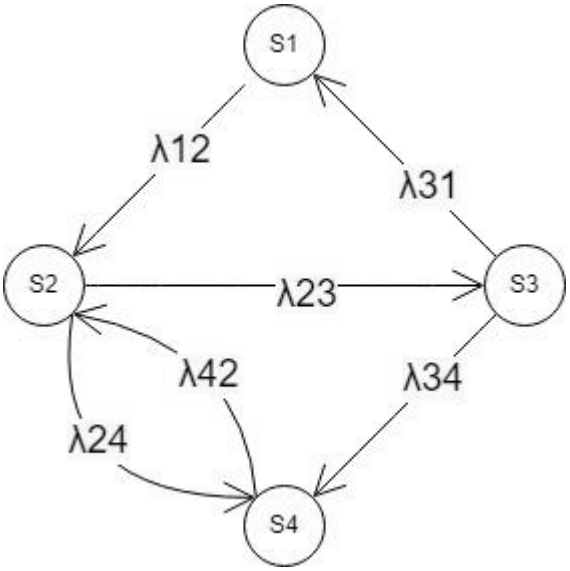
- в левой части каждого из уравнений стоит производная вероятности i-ого состояния;
- в правой части содержится столько членов, сколько стрелок связано с данным состоянием;
- если стрелка направлена из состояния, соответствующий член имеет знак «минус», если в состояние — знак «плюс»;
- каждый член равен произведению интенсивности, соответствующей данной стрелке, и вероятности того состояния, из которого выходит стрелка.

Если в уравнениях Колмагорова приравнять производные к нулю, то получим систему уравнений, описывающих стационарный режим.

Для поиска решений необходимо добавить уравнение нормировки:

$$P_1 + P_2 + \cdots + P_k + \cdots + P_n = 1 \qquad (2)$$

Пример



$$P_1(t + \Delta t) = P_1(t)(1 - \lambda_{12}.\Delta t) + P_3(t).\lambda_{31}.\Delta t$$

$$P_1'(t) = -\lambda_{12}.P_1(t) + \lambda_{31}.P_3(t)$$

$$P_2'(t) = -P_2(t).\lambda_{23} - P_2(t).\lambda_{24} + P_1(t).\lambda_{12} + P_4(t).\lambda_{42}$$

$$P_3'(t) = -P_3(t).\lambda_{31} - P_3(t).\lambda_{34} + P_2(t).\lambda_{23}$$

$$P_4'(t) = -P_4(t).\lambda_{42} + P_2(t).\lambda_{24} + P_3(t).\lambda_{34}$$

II. Результаты

Эксперимент I (Система имеет 4 состояния)

		S1	S2	S3	S4
Данные	S1	**0**	**2**	**0**	**0**
	S2	**0**	**0**	**2**	**3**
	S3	**3**	**0**	**0**	**1**
	S4	**0**	**4**	**0**	**0**
Результаты	P	**0.24**	**0.32**	**0.16**	**0.28**
	T1	**2.199**	**2.899**	**3.074**	**3.376**
	T2	**1.612**	**0958**	**0.536**	**1.359**

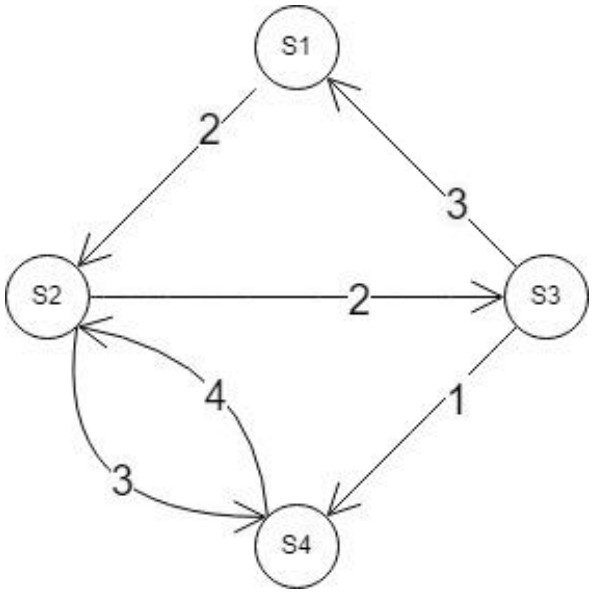


Рисунок 1. граф связей и интенсивностей системы Эксперимент I

T1 - Время стабилизации, при начальных условиях $P_1 = 1$, $P_{2,3,4} = 0$

T2 - Время стабилизации, при начальных условиях $P_{1,2,3,4} = 1/4$

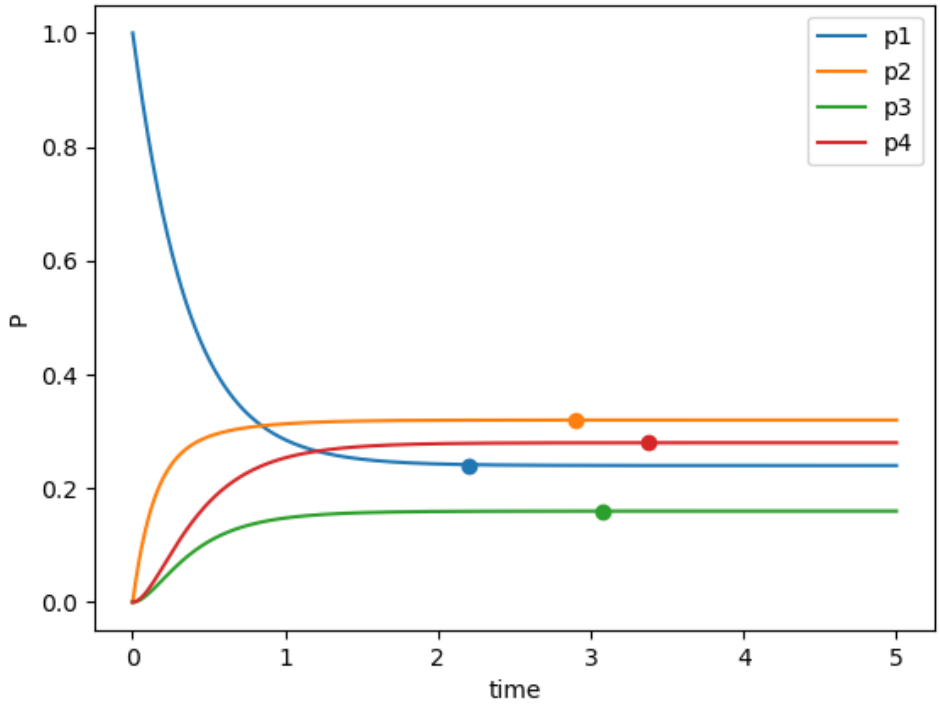


Рисунок 2. графики вероятностей состояний как функции времени, при начальных условиях $P_1 = 1$, $P_{2,3,4} = 0$

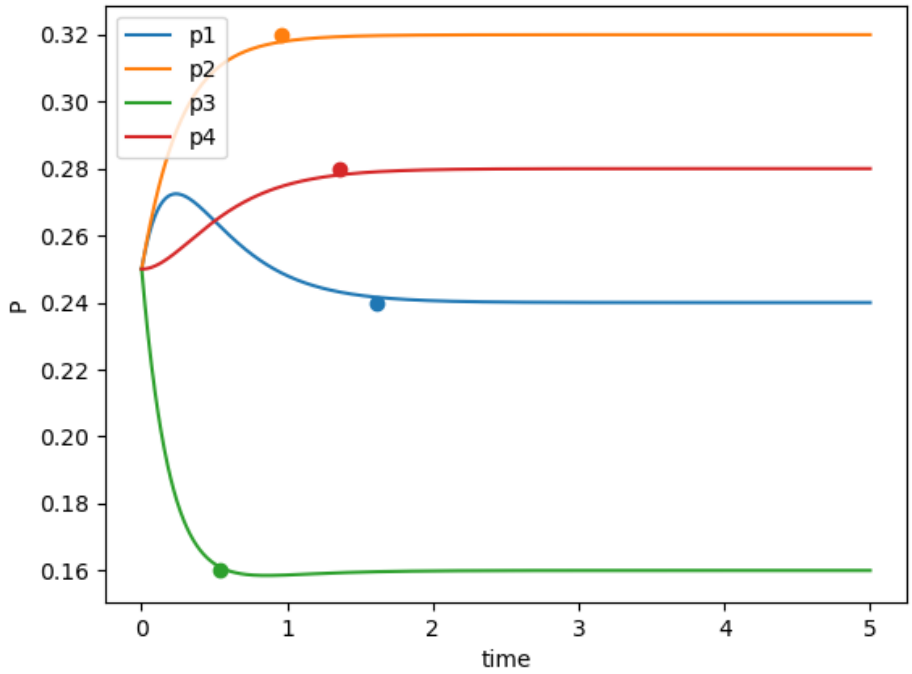


Рисунок 3. графики вероятностей состояний как функции времени, при начальных условиях $P_{1,2,3,4} = 1/4$

Эксперимент II (Система имеет 5 состояния)

		S1	S2	S3	S4	S5
Данные	S1	**0**	**1**	**1**	**2**	**0**
	S2	**0**	**0**	**0**	**0**	**0.5**
	S3	**0**	**2**	**0**	**1**	**0**
	S4	**0**	**0**	**3**	**0**	**1.5**
	S5	**2**	**0**	**0**	**2**	**0**
Результаты	P	**0.057**	**0.603**	**0.122**	**0.103**	**0.114**
	T1	**1.494**	**4.734**	**3.334**	**2.880**	**3.292**
	T2	**1.525**	**4.631**	**3.221**	**3.773**	**0.911**

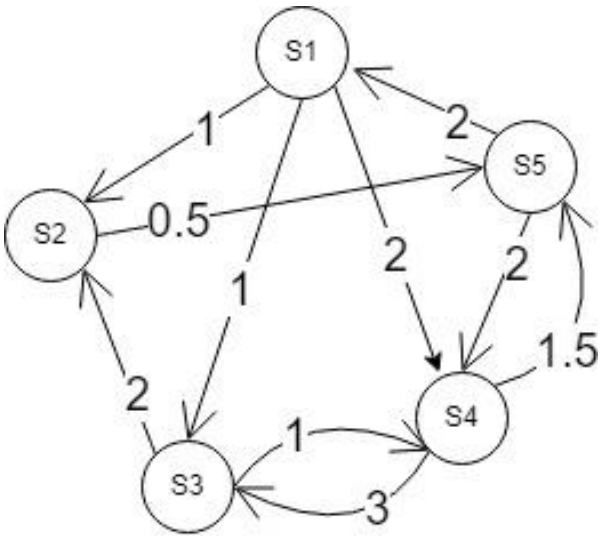


Рисунок 4. граф связей и интенсивностей системы Эксперимент II

T1 - Время стабилизации, при начальных условиях $P_1 = 1$, $P_{2,3,4,5} = 0$

T2 - Время стабилизации, при начальных условиях $P_{1,2,3,4,5} = 1/5$

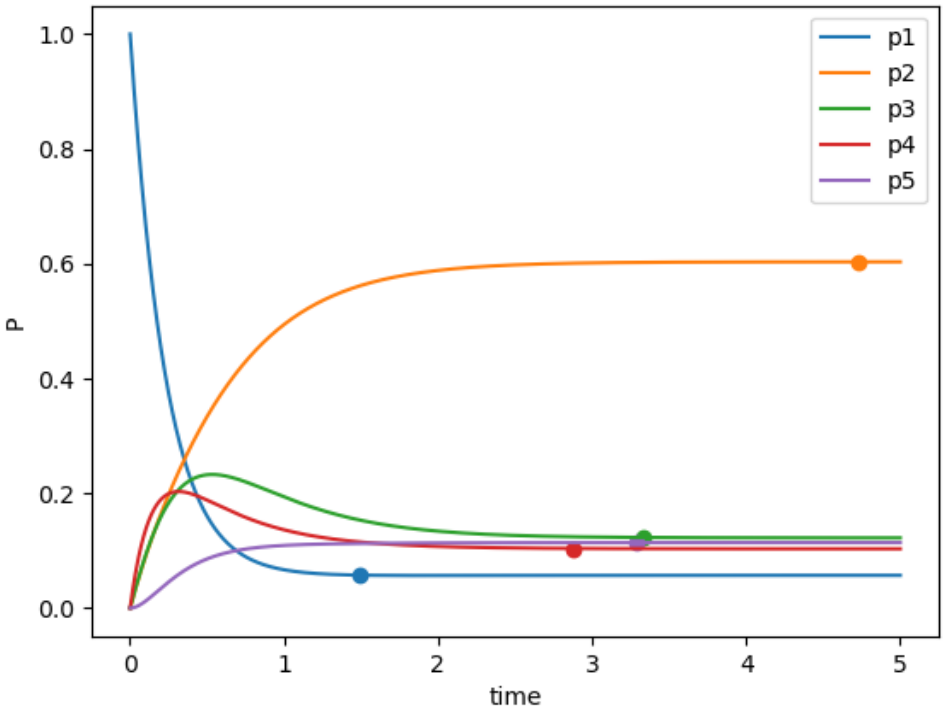


Рисунок 5. графики вероятностей состояний как функции времени, при начальных условиях $P_1 = 1$, $P_{2,3,4,5} = 0$

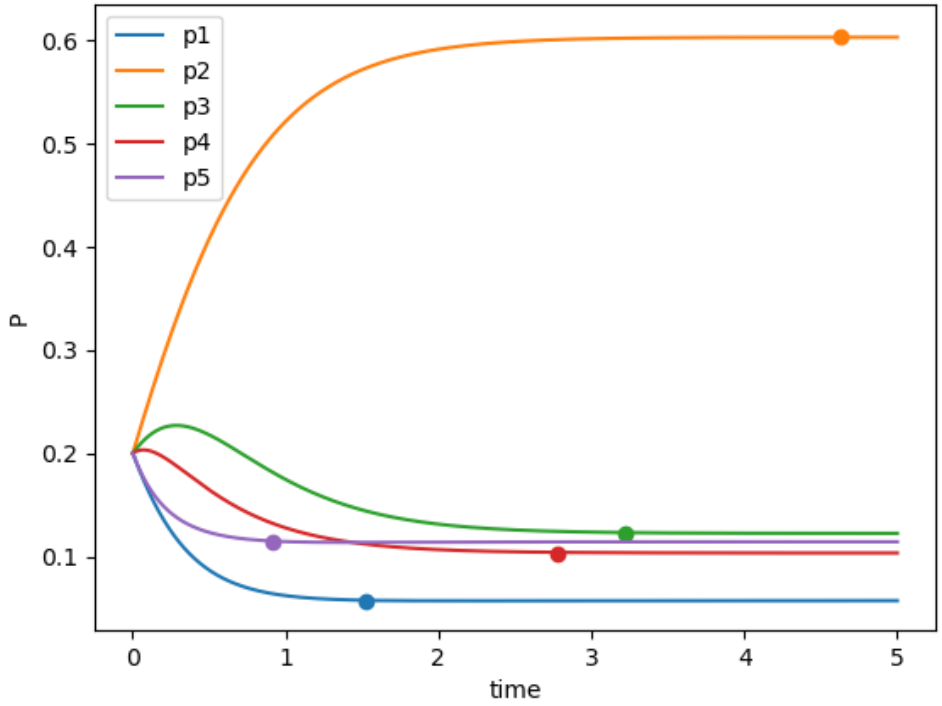


Рисунок 6. графики вероятностей состояний как функции времени, при начальных условиях $P_{1,2,3,4,5} = 1/5$

III. Код Программы

Main.py

```
from math import fabs
import random
import matplotlib.pyplot as plt
import numpy

PRECISION = 5
TIME_DELTA = 1e-3
MAGIC_NUM = 10

def dps(matrix, P):
    n = len(matrix)
    res = [sum(
            [
                P[j] * (-sum(matrix[i]) + matrix[i][i]) if i == j else
P[j] * matrix[j][i] for j in range(n)
            ]
        )
        for i in range(n)]

    return [i * TIME_DELTA for i in res]

def calcStabilizationTimes(matrix, start_P, limit_P):
    n = len(matrix)
    current_time = 0
    current_P = start_P.copy()
    stabilizationTimes = [0 for i in range(n)]

    total_lambda_sum = sum([sum(i) for i in matrix]) * MAGIC_NUM
    cool_eps = [p/total_lambda_sum for p in limit_P]

    while not all(stabilizationTimes):
        curr_dps = dps(matrix, current_P)
        for i in range(n):
            if (not stabilizationTimes[i] and curr_dps[i] <= 1e-7 and
                    abs(current_P[i] - limit_P[i]) <= cool_eps[i]):
                stabilizationTimes[i] = current_time
            current_P[i] += curr_dps[i]

        current_time += TIME_DELTA

    return stabilizationTimes
```

```
def calcPOverTime(matrix, start_P, end_time):
    n = len(matrix)
    current_time = 0
    current_P = start_P.copy()

    POverTime = []
    times = []

    while current_time < end_time:
        POverTime.append(current_P.copy())
        curr_dps = dps(matrix, current_P)
        for i in range(n):
            current_P[i] += curr_dps[i]

        current_time += TIME_DELTA

        times.append(current_time)

    return times, POverTime


def buildCoeffMatrix(matrix):
    matrix = numpy.array(matrix)
    n = len(matrix)
    res = numpy.zeros((n, n))

    for state in range(n - 1):
        for col in range(n):
            res[state, state] -= matrix[state, col]
        for row in range(n):
            res[state, row] += matrix[row, state]

    for state in range(n):
        res[n - 1, state] = 1

    return res


def buildAugmentationMatrix(count):
    res = [0 for i in range(count)]
    res[count - 1] = 1
    return numpy.array(res)


def solve(matrix):
    coeffMatrix = buildCoeffMatrix(matrix)
    augmentationMatrix = buildAugmentationMatrix(len(matrix))
    return numpy.linalg.solve(coeffMatrix, augmentationMatrix)

def graphPOverTime(P, stabilizationTime, times, POverTime):
    for i_node in range(len(POverTime[0])):
        plt.plot(times, [i[i_node] for i in POverTime])
```

```
            plt.scatter(stabilizationTime[i_node], P[i_node])

        plt.legend(['p{}'.format(i+1) for i in range(len(P))])
        plt.xlabel('time')
        plt.ylabel('P')
        plt.show()


def random_matrix(size):
    return [
        [round(random.random(), PRECISION) if i != j else 0.0 for j in
range(size)]
        for i in range(size)
    ]


def output(title, caption, data):
    print(title)
    for i in range(len(data)):
        print(caption + str(i), round(fabs(data[i]), PRECISION))
    print()


def getPreDefineI(i):
    if i == 3:
        return [[0, 2, 0],
                [1, 0, 0],
                [0, 1, 1]]
    if i == 4:
        return [[0, 2, 0, 0],
                [0, 0, 2, 3],
                [3, 0, 0, 1],
                [0, 4, 0, 0]]
    elif i == 5:
        return [[0, 1, 1, 2, 0],
                [0, 0, 0, 0, 0.5],
                [0, 2, 0, 1, 0],
                [0, 0, 3, 0, 1.5],
                [2, 0, 0, 2, 0]]


def getStartP(n, all_equal=True):
    if all_equal:
        return [1/n] * n
    else:
        res = [0] * n
        res[0] = 1
        return res


if __name__ == '__main__':
    n = 5
```

```
#I = random_matrix(n)
I = getPreDefineI(n)

start_P = getStartP(n, True)

P = solve(I)
output('P:', 'p', P)

stabilizationTime = calcStabilizationTimes(I, start_P, P)
times, POverTime = calcPOverTime(I, start_P, 5)
output('T:', 't', stabilizationTime)

graphPOverTime(P, stabilizationTime, times, POverTime)
```