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

Название Цепи Маркова

Дисциплина Моделирование

Студент Прохорова Л. А.

Группа ИУ7-73Б

Оценка (баллы) _____

Преподаватель Рудаков И. В.

Москва — 2021 г.

1 Задание

Написать программу которая позволяет определить время пребывания сложной системы в каждом из состояний. Количество состояний не больше 10. при t $\to \infty$. Граф задается матрицей.

2 Теоретическая часть

Случайный процесс, протекающий в некоторой 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 и не зависит от того, когда и каким образом система пришла в этот состояние(то есть как развивался процесс в прошлом). В природе нет таких процессов, но существует ряд процессов, которые могут некоторыми методами быть сведены к марковским процессам. Вероятностью i-го состояния называется вероятность $p_i(t)$ того, что в момент t система будет находиться в состоянии S_i Для любого момента t сумма вероятностей всех состояний равна единице.

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 (i-го состояния).

Рассмотрим методику введения уравнения Колмогорова для функционирования системы.

Пусть система имеет 4 возможных состояния. Эти все состояния свя-

заны между собой интенсивностью.

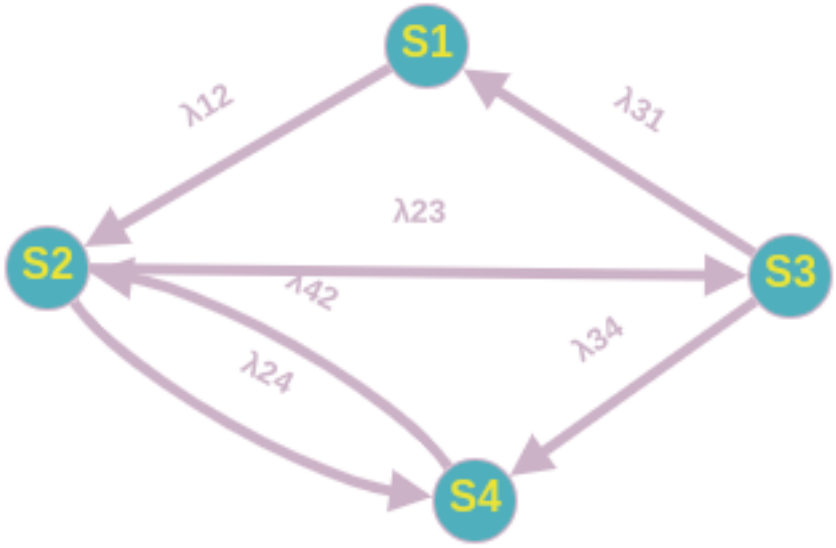

Рисунок 1 – Граф состояний.

$$\begin{cases} p_1^{'}(t) = -\lambda_{12}p_1(t) + \lambda_{31}p_3(t) \\ p_2^{'}(t) = -\lambda_{24}p_2(t) - \lambda_{23}p_2(t) + \lambda_{42}p_4(t) + \lambda_{12}p_1(t) \\ p_3^{'}(t) = -\lambda_{31}p_3(t) - \lambda_{34}p_3(t) + \lambda_{23}p_2(t) \\ p_4^{'}(t) = -\lambda_{42}p_4(t) + \lambda_{24}p_2(t) + \lambda_{34}p_3(t) \end{cases} \tag{1}$$

Для получения предельных вероятностей, то есть вероятностей в стационарном режиме работы при $t \to \infty$, необходимо приравнять левые части уравнений к нулю. Таким образом получается система линейных уравнений. Для решения полученной системы необходимо добавить условие нормировки $\sum_i^n p_i = 1$

После того, как предельные вероятности будут найдены, необходимо найти время. Для этого необходимо с интервалом Δt находить каждую вероятность в момент времени $t+\Delta t$. На каждом шаге необходимо вычислять приращения для каждой вероятности (как функции):

$$\Delta p_i = p_i^{'}(t) * \Delta t \tag{2}$$

Когда найденная вероятность будет равна соответствующей предыдущей с точностью до заданной погрешности, тогда можно завершить вычисления.

3 Результаты работы программы

На рисунках 2, 3и 4 представлено заполнение матрицы и результат для 4 состояний(ввод интенсивности производит пользователь).

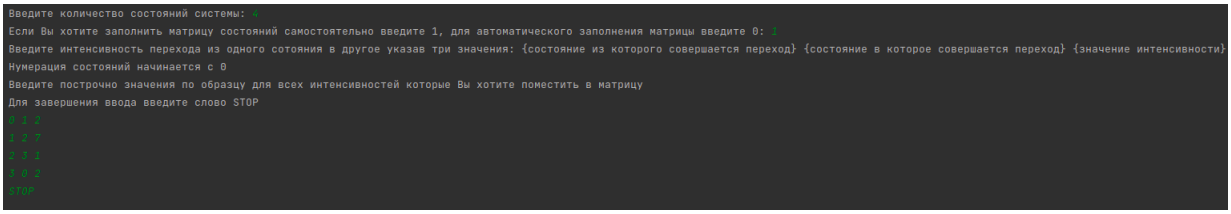

Рисунок 2 – Заполнение матрицы

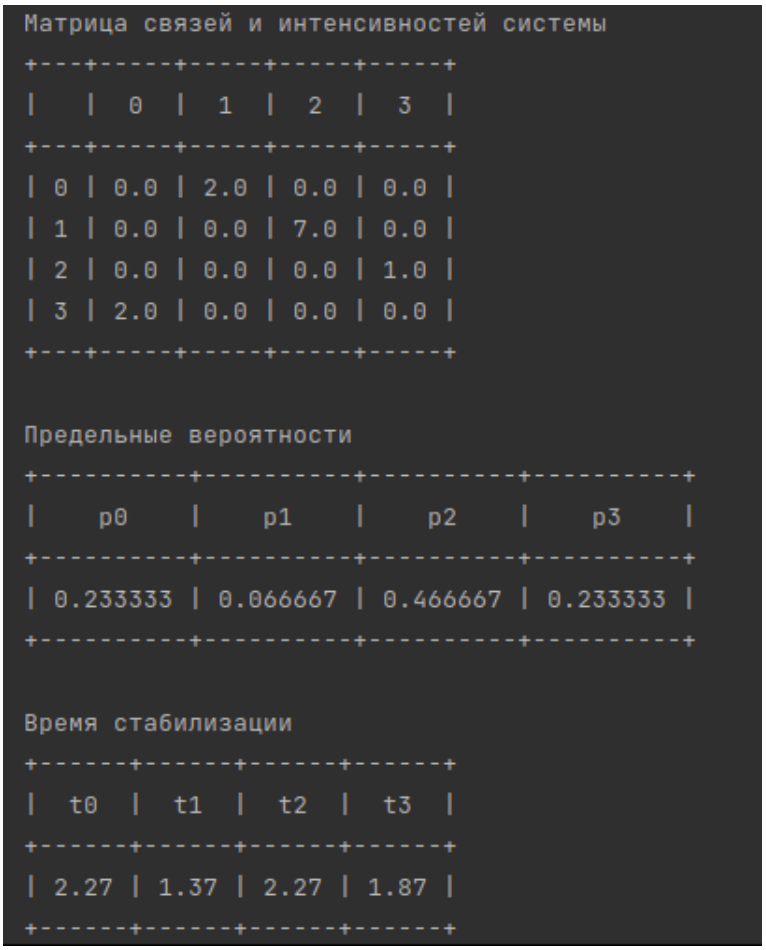

Рисунок 3 – Матрица и результат

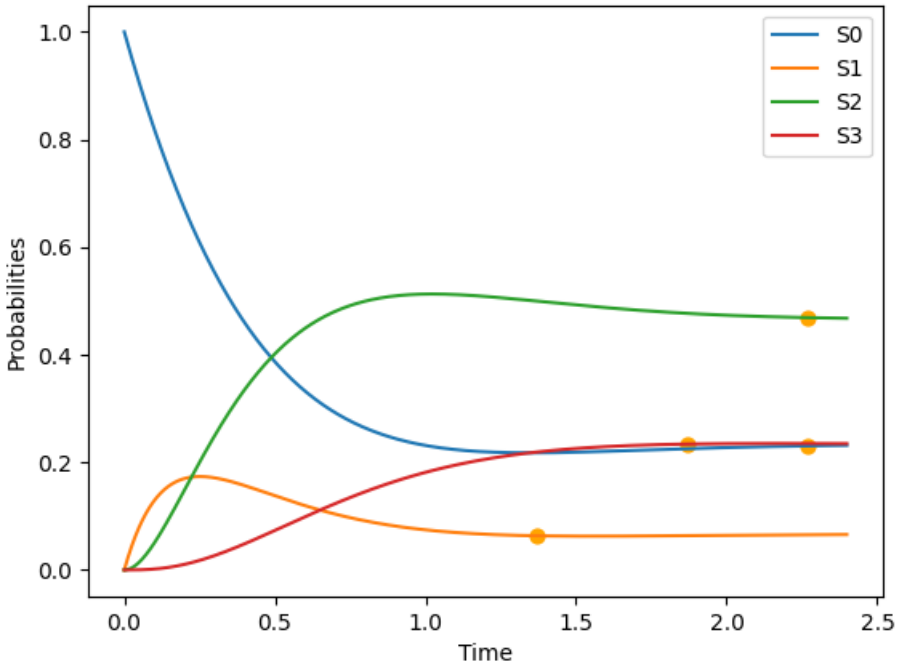

Рисунок 4 – График вероятностей состояний как функции времени

На рисунках 5 и 6 представлено заполнение матрицы и результат для 6 состояний(заполнение матрицы происходит автоматически).

```
Введите количество состояний системы: 6
Если Вы хотите заполнить матрицу состояний самостоятельно введите 1, для автоматического заполнения матрицы введите 0: 0

Матрица связей и интенсивностей системы
+---+--------+--------+--------+--------+--------+--------+
|   |   0    |   1    |   2    |   3    |   4    |   5    |
+---+--------+--------+--------+--------+--------+--------+
| 0 |  0.0   | 0.8173 | 0.6678 | 0.5218 | 0.1345 | 0.0449 |
| 1 | 0.8029 |  0.0   | 0.421  | 0.632  | 0.7914 | 0.0797 |
| 2 | 0.1295 | 0.9984 |  0.0   | 0.1756 | 0.9082 | 0.9031 |
| 3 | 0.5347 | 0.0321 | 0.4653 |  0.0   | 0.1245 | 0.8262 |
| 4 | 0.1583 | 0.8224 | 0.6012 | 0.8334 |  0.0   | 0.8538 |
| 5 | 0.3635 | 0.1809 | 0.3542 | 0.4267 | 0.5262 |  0.0   |
+---+--------+--------+--------+--------+--------+--------+

Предельные вероятности
+----------+----------+----------+----------+----------+----------+
|    p0    |    p1    |    p2    |    p3    |    p4    |    p5    |
+----------+----------+----------+----------+----------+----------+
| 0.160034 | 0.152288 | 0.134906 | 0.203392 | 0.124821 | 0.224559 |
+----------+----------+----------+----------+----------+----------+

Время стабилизации
+------+------+------+------+------+------+
|  t0  |  t1  |  t2  |  t3  |  t4  |  t5  |
+------+------+------+------+------+------+
| 2.17 | 1.97 | 0.87 | 1.57 | 1.47 | 2.37 |
+------+------+------+------+------+------+
```

Рисунок 5 – Заполнение матрицы и результаты

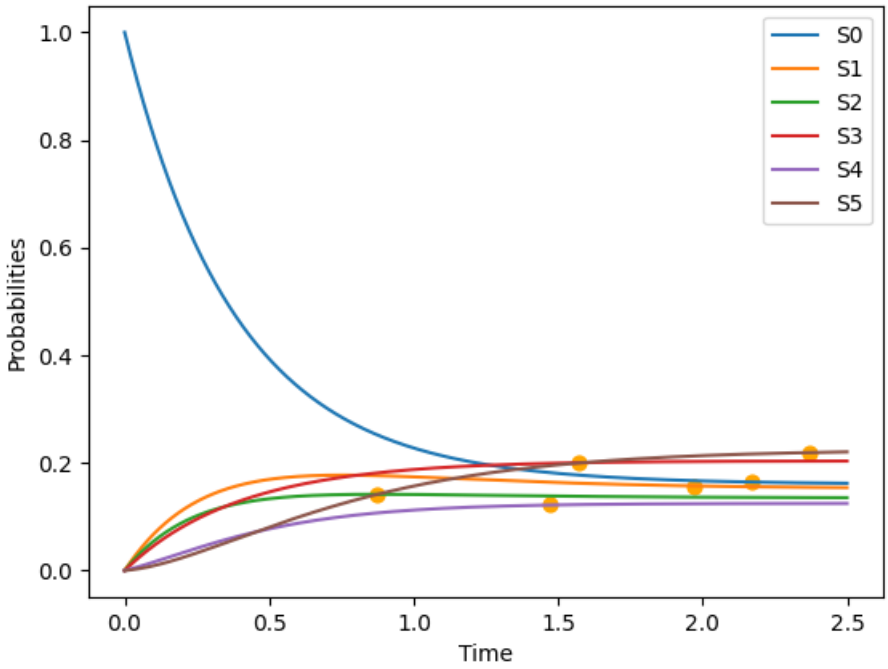

Рисунок 6 – График вероятностей состояний как функции времени

4 Код программы

Программа разработана в интегрированной среде разработки для языка программирования Python - PyCharm. В листинге 4 приведена реализация лабораторной работы.

```
import random
import numpy
from prettytable import PrettyTable
import matplotlib.pyplot as plt
TIME_DELTA = 1e-3
EPS = 1e-5

def dp(matrix, probabilities):
    res = []
    n = len(matrix)
    for i in range(n):
        summ = 0
        for j in range(n):
            if i == j:
                sum_i = 0
```

```
                for t in range(n):
                    sum_i += matrix[i][t]

                summ += probabilities[j] * (-1 * sum_i + matrix[i][i])
            else:
                summ += probabilities[j] * matrix[j][i]
        res.append(TIME_DELTA * summ)
    return res

def get_stabilization_times(matrix, start_probabilities):
    n = len(matrix)
    current_time = 0
    current_probabilities = start_probabilities.copy()
    stabilization_times = [0 for i in range(n)]
    stabilization_p = [0 for i in range(n)]
    prev_probabilities = []
    for i in range(n):
        prev_probabilities.append([])
    x = []
    counter = 0
    prev_dp = dp(matrix, current_probabilities)
    while not all(stabilization_times):
        while counter < 100:
            curr_dp = dp(matrix, current_probabilities)
            for i in range(n):
                prev_probabilities[i].append(current_probabilities[i])
                current_probabilities[i] += curr_dp[i]
            counter += 1
            x.append(current_time)
            current_time += TIME_DELTA
        for i in range(n):
            if not stabilization_times[i] and abs(prev_dp[i] - curr_dp[i])
    < EPS and abs(curr_dp[i]) < EPS:
                stabilization_times[i] = current_time - TIME_DELTA * 30
                stabilization_p[i] = current_probabilities[i]
        counter = 0
        prev_dp = curr_dp

    counter = 0
    while counter < 100:
```

```
54          curr_dp = dp(matrix, current_probabilities)
55          for i in range(n):
56              prev_probabilities[i].append(current_probabilities[i])
57              current_probabilities[i] += curr_dp[i]
58          counter += 1
59          x.append(current_time)
60          current_time += TIME_DELTA
61      fig, ax = plt.subplots()
62
63      for i in range(n):
64          ax.plot(x, prev_probabilities[i], label = 'S' + str(i))
65          ax.scatter(stabilization_times[i], stabilization_p[i], color='
    orange', s=40, marker='o')
66      ax.legend()
67      ax.set_xlabel('Time')
68      ax.set_ylabel('Probabilities')
69      plt.show()
70      return stabilization_times
71
72
73  def solve(matrix):
74      matrix = numpy.array(matrix)
75      n = len(matrix)
76      coeff_matrix = numpy.zeros((n, n))
77
78      for state in range(n - 1):
79          for col in range(n):
80              coeff_matrix[state, state] -= matrix[state, col]
81          for row in range(n):
82              coeff_matrix[state, row] += matrix[row, state]
83
84      for state in range(n):
85          coeff_matrix[n - 1, state] = 1
86
87      res = [0 for i in range(n)]
88      res[n - 1] = 1
89      augmentation_matrix = numpy.array(res)
90
91      return numpy.linalg.solve(coeff_matrix, augmentation_matrix)
92
```

```
93 if __name__ == '__main__':
94     n = int(input("Введите количество состояний системы: "))
95     if n <= 0 or n > 10:
96         print("Некорректное количество состояний")
97         exit(1)
98     matrix = []
99     for i in range(n):
100         matrix.append([])
101         for j in range(n):
102             matrix[i].append(0.0)
103 
104     flag = input("Если Вы хотите заполнить матрицу состояний самостоятельно введите
    1, "
105                  "для автоматического заполнения матрицы введите 0: ")
106     if flag == "0":
107         for i in range(n):
108             for j in range(n):
109                 if i != j:
110                     matrix[i][j] = round(random.random(), 4)
111 
112     elif flag == "1":
113         print("Введите интенсивность перехода из одного сотояния в другое указав
    три значения: состояние{ из которого совершается переход} состояние{ в которое
    совершается переход} значение{ интенсивности}")
114         print("Нумерация состояний начинается с 0")
115         print("Введите построчно значения по образцу для всех интенсивностей
    которые Вы хотите поместить в матрицу")
116         print("Для завершения ввода введите слово STOP")
117 
118         while(1):
119             s = input()
120             if s == "STOP":
121                 break
122             i, j, lmbda = map(float, s.split())
123             if i == j:
124                 print("Переход в одно и то же состояние невозможен")
125             elif (i < 0 or i >= n or j < 0 or j >= n):
126                 print("Несуществует таких состояний")
127             else:
128                 matrix[int(i)][int(j)] = lmbda
```

```
    else:
        print("Некорректный ввод")
        exit(1)

    print("\Матрицаn связей и интенсивностей системы")
    table = PrettyTable()
    names = [""]
    for i in range(n):
        names.append(str(i))
    table.field_names = names

    for i in range(n):
        tmp = [i]
        tmp.extend(item for item in matrix[i])
        table.add_row(tmp)
    print(table)

    # начальные значения для dp
    start_probabilities = [0] * n
    start_probabilities[0] = 1

    # вычисление предельных вероятностей
    probability = solve(matrix)
    print("\Предельныеn вероятности")

    table_probability = PrettyTable()
    names = []
    for i in range(n):
        names.append("p" + str(i))
    table_probability.field_names = names

    tmp = []
    tmp.extend(round(item, 6) for item in probability)
    table_probability.add_row(tmp)
    print(table_probability)

    # поиск времени стабилизации
    stabilization_time = get_stabilization_times(matrix,
start_probabilities)

```

```
168     print("\Времяn стабилизации")
169
170     table_stabilization = PrettyTable()
171     names = []
172     for i in range(n):
173         names.append("t" + str(i))
174     table_stabilization.field_names = names
175
176     tmp = []
177     tmp.extend(round(item, 6) for item in stabilization_time)
178     table_stabilization.add_row(tmp)
179     print(table_stabilization)
```