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 Виноградов А. О.

Группа ИУ7-76Б

Оценка (баллы)

Преподаватель Рудаков И. В.

Москва — 2023 г.

1 Теоретическая часть

Случайный процесс называется марковским, если он обладает следующим свойством: для каждого момента 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.

Вероятностью i-го состояния называется вероятность $p_i(t)$ того, что в момент t система будет находится в состоянии S_i. Также для любого времени t выполняется условие нормировки — сумма вероятностей всех состоянии равна единице.

Стационарное распределение цепи Маркова — распределение вероятности, которое не меняется с течением времени.

Предельная вероятность состояния показывает среднее время пребывания системы в этом состоянии. Для нахождения таких вероятностей используют уравнения Колмогорова.

Уравнение Колмогорова строится по следующему правилу: в левой части каждого уравнения стоит производная вероятности состояния, а правая часть содержит столько членов сколько стрелок связано с данным состоянием.

Если стрелка направлена из состояния, то соответствующий член имеет знак "− в состояние — "+".

Каждый член равен произведению плотности вероятности перехода соответствующий данной стрелке, умноженной на вероятность того состояния, из которого исходит стрелка.

Получаем систему уравнений вида:

$$\begin{cases} p_1'(t) = -(\lambda_{11} + ... + \lambda_{1n})p_1 + \lambda_{21} * p_2 + ... + \lambda_{n1} * p_n \\ p_2'(t) = -(\lambda_{21} + ... + \lambda_{2n})p_2 + \lambda_{12} * p_1 + ... + \lambda_{n2} * p_n \\ ... \\ p_n'(t) = -(\lambda_{n1} + ... + \lambda_n n)p_n + \lambda_{1n} * p_1 + ... + \lambda_{nn} * p_n. \end{cases} \tag{1.1}$$

В стационарном случае ($p_i'(t) = 0, i = \overline{1,n}$) левые системы уравнений принимают значение 0. Получаем СЛАУ, решение которой дает нам искомые предельные вероятности состояний системы.

Интегрирование системы (1.1) дает искомые вероятности системы как функции времени. График зависимости вероятности пребывания системы в состоянии p_i колеблется, а затем в определенный момент $t_i!$ принимает стабильное значение $p_i!$. Это время $t_i!$ до стабилизации вычисляется в реализованной программе для каждого $i = \overline{1,n}$.

2 Результат

На рисунке 2.1 приведен пример работы программы.

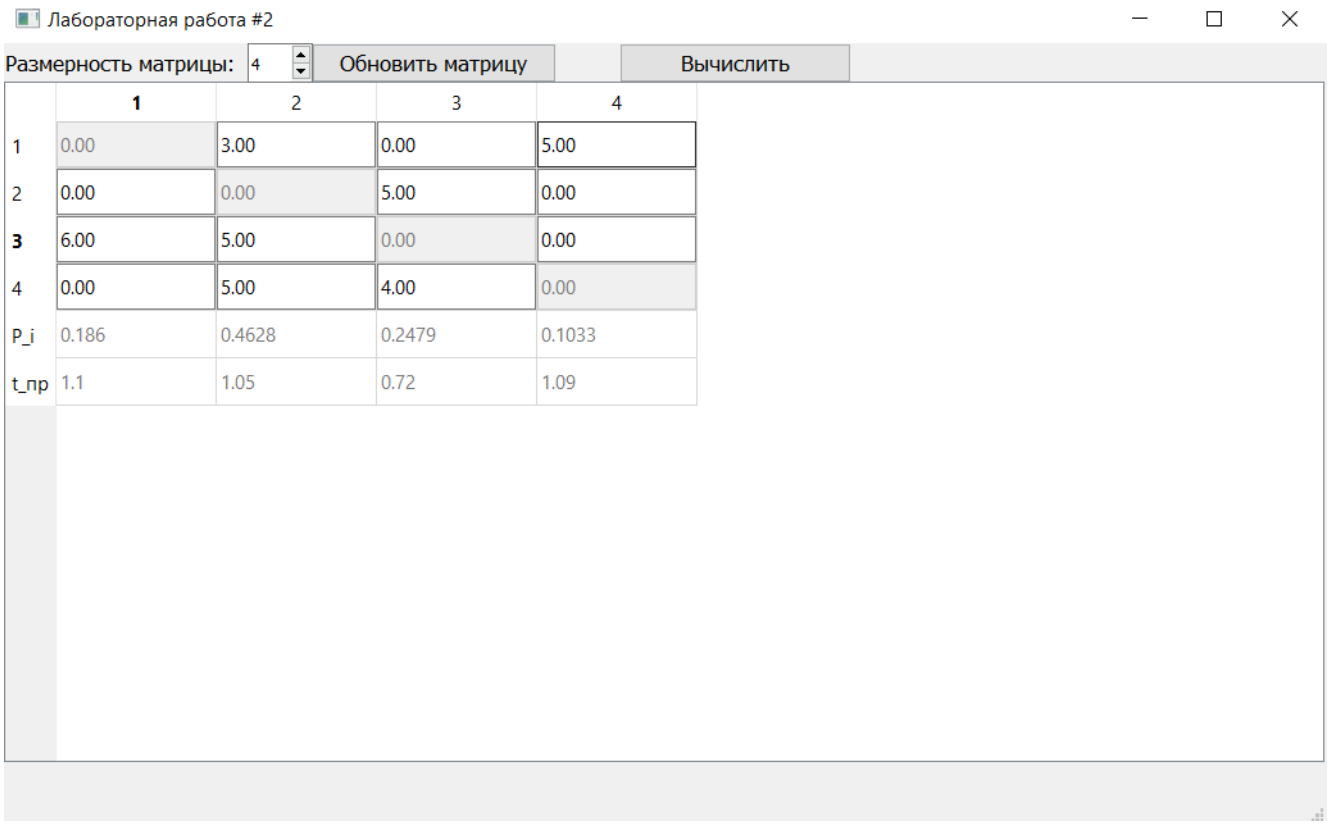

Рисунок 2.1 – Пример работы программы

В листингах 2.1 – 2.2 приведена реализация вычислений коэффициентов уравнений Колмогорова, вычисления предельных вероятностей и времени до стабилизации.

Листинг 2.1 – Реализация программных вычислений

```
1 def kolm_coef(matrix):
2     n = len(matrix)
3
4     koef = [[0 for _ in range(n)] for i in range(n)]
```

Листинг 2.2 – Реализация программных вычислений (продолжение)

```
    for i in range(len(matrix)):
        for j in range(len(matrix)):
            if i != j:
                koef[i][j] = matrix[j][i]
                koef[i][i] -= matrix[i][j]
    return koef

def limit_prob(matrix):
    n = len(matrix)
    koef = kolm_coef(matrix)
    koef[0] = [1] * n

    free = [1] + [0] * (n - 1)

    return linalg.solve(koef, free)

def f(y, t, koef):
    sp = [0 for i in range(len(y))]
    for i in range(len(y)):
        for j in range(len(y)):
            sp[i] += koef[i][j] * y[j]
    return sp


def stable_time(matrix, probs):
    n = len(probs)
    time = arange(0, max_time, dt)

    start = [1] + [0] * (n - 1)
    koef = kolm_coef(matrix)
    probabilities = transpose(odeint(f, start, time, args =
       (koef,)))

    time_to_stable = [-1] * n

    for i in range(n):
        for j, cur_prob in enumerate(probabilities[i]):
            if abs(probs[i] - cur_prob) < eps:
                time_to_stable[i] = time[j]
                break

    return time_to_stable
```