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 (ИУ)»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии (ИУ7)»

# ОТЧЕТ

по Лабораторной работе №2

по курсу «Моделирование»

на тему: «Марковские процессы»

Студент ИУ7-73Б (Группа) _______ (Подпись, дата) К.Э. Ковалец (И. О. Фамилия)

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

2022 г.

# Содержание

# 1 Задание

Разработать графический интерфейс, который позволяет по заданной матрице интенсивностей перехода состояний определить время пребывания системы в каждом состоянии в установившемся режиме работы системы. Для каждого состояния также требуется рассчитать предельную вероятность. Количество состояний не более десяти.

# 2 Теоретическая часть

## 2.1 Марковский процесс

Случайный процесс называется марковским процессом, если для каждого момента времени $t$ вероятность любого состояния системы в будущем зависит только от ее состояния в настоящем и не зависит от того, как система пришла в это состояние.

## 2.2 Предельная вероятность

Для определения предельной вероятности необходимо решить систему уравнений Колмагорва, в которой все производные приравниваются к нулю, а одно из уравнений заменяется на условие нормировки:

$$\sum_{j=1}^{n} p_j(t) = 1. \tag{2.1}$$

## 2.3 Точки стабилизации состояния системы

Для определения точек стабилизации состояния системы нужно определить вероятности нахождения в определённых состояниях с некоторым малым шагом $\Delta t$. В тот момент, когда разница между вычисленной на данном шаге вероятностью и предельной вероятности будет достаточно мала ($< EPS$), то точка стабилизации считается найденной.

# 3 Результаты работы

## 3.1 Листинги программы

В листинге 3.1 представлен класс $MarkovChains$, отвечающий за определение времени пребывания системы в каждом состоянии в установившемся режиме работы системы и за рассчет предельных вероятностей.

Листинг 3.1 — class MarkovChains

```python
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

EPS = 1e-3


class MarkovChains():
    matrix: list
    matrixSize: int

    initProbs: list
    dt: float


    def __init__(self, matrix: int, matrixSize: int, dt: float):
        self.matrix = matrix
        self.matrixSize = matrixSize
        self.initProbs = self.createInitProbabilities(matrixSize)
        self.dt = dt


    def createInitProbabilities(self, arraySize):
        return [1 if i == 0 else 0 for i in range(arraySize)]


    def getProbabilities(self):
        freeMembers = [0 for _ in range(self.matrixSize - 1)]
        freeMembers.append(1)

        matrixCoeffs = [
            [
                -sum(self.matrix[i]) + self.matrix[i][j] if j == i else
                ↪  self.matrix[j][i]
                for j in range(self.matrixSize)
            ]
```

```python
            for i in range(self.matrixSize - 1)
        ]
        matrixCoeffs.append([1 for _ in range(self.matrixSize)])

        # Стабильное состояние
        probsSteady = np.linalg.solve(matrixCoeffs, freeMembers)

        return probsSteady


    def solveOde(self, initProbs: list, _, matrixCoeffs: list):
        dydt = [0 for _ in range(self.matrixSize)]

        for i in range(self.matrixSize):
            dydt[i] = sum(initProbs[j] * matrixCoeffs[i][j]
                for j in range(self.matrixSize))

        return dydt


    def getTimes(self, probsSteady: list, buildGraph: bool):
        matrixCoeffs = [
            [
                -sum(self.matrix[i]) + self.matrix[i][j] if j == i else
                ↪  self.matrix[j][i]
                for j in range(self.matrixSize)
            ]
            for i in range(self.matrixSize)
        ]

        times = np.arange(0, 20, self.dt)

        resOde = odeint(self.solveOde, self.initProbs, times, args=(matrixCoeffs,))
        resOde = np.transpose(resOde)

        timesSteady = list()

        for i in range(self.matrixSize):
            if buildGraph:
                plt.plot(times, resOde[i], label = "p{}".format(i))

            for j in range(len(resOde[i]) - 1, -1, -1):
                if abs(probsSteady[i] - resOde[i][j]) > EPS:
                    # Времена достижения стабильного состояния
                    timesSteady.append(times[j])
                    break

```

```
        if buildGraph:
            plt.legend()
            plt.grid()
            plt.show()

        return timesSteady


    def solve(self, buildGraph: bool):
        probsSteady = self.getProbabilities()
        timesSteady = self.getTimes(probsSteady, buildGraph)

        return [probsSteady, timesSteady]
```

## 3.2 Демонстрация работы программы

На рисунках 3.1 - 3.4 представлены примеры работы программы.

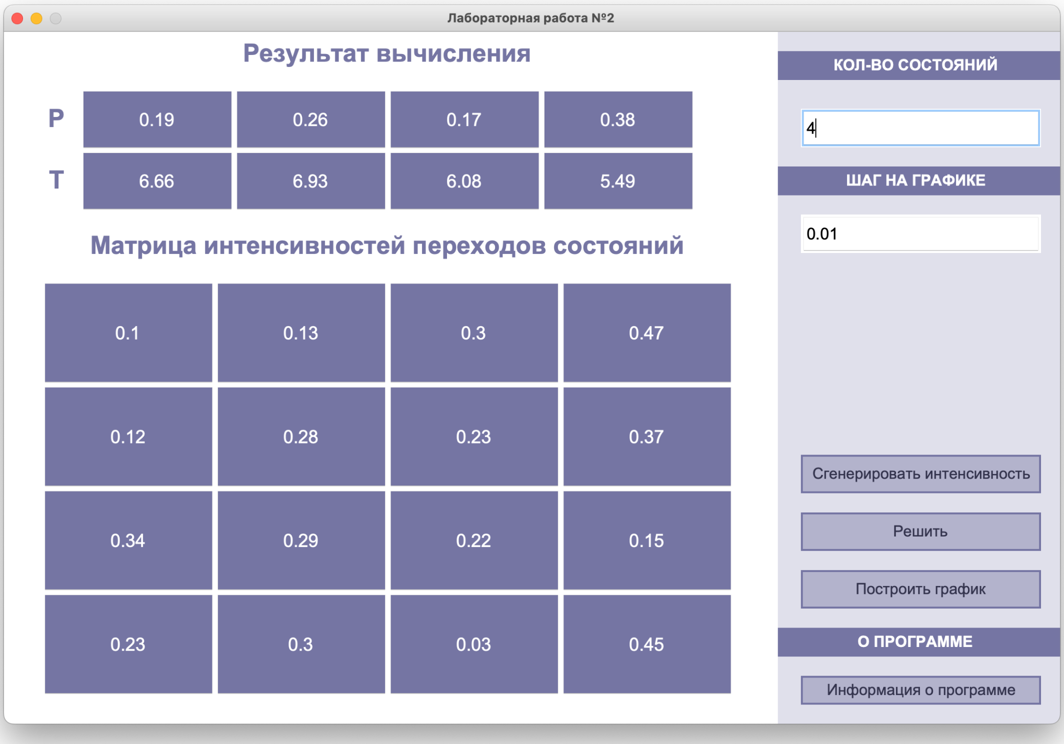


Рисунок 3.1 – Система из 4 состояний

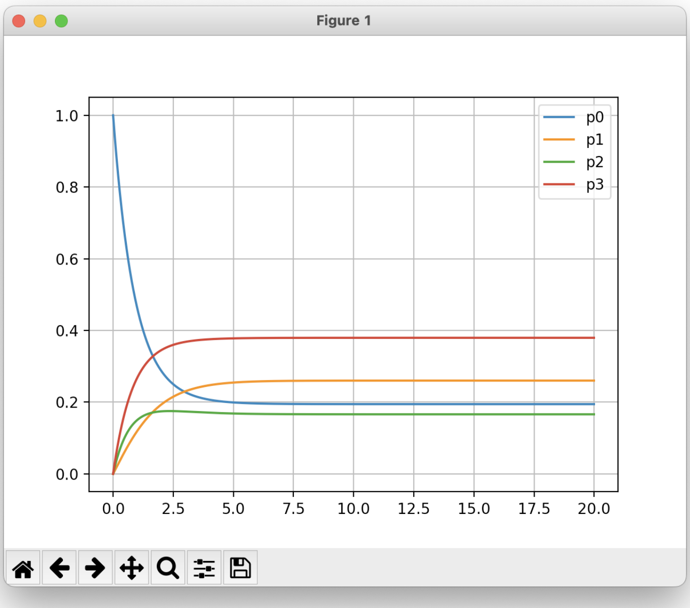


Рисунок 3.2 – График вероятности от времени для системы из 4 состояний

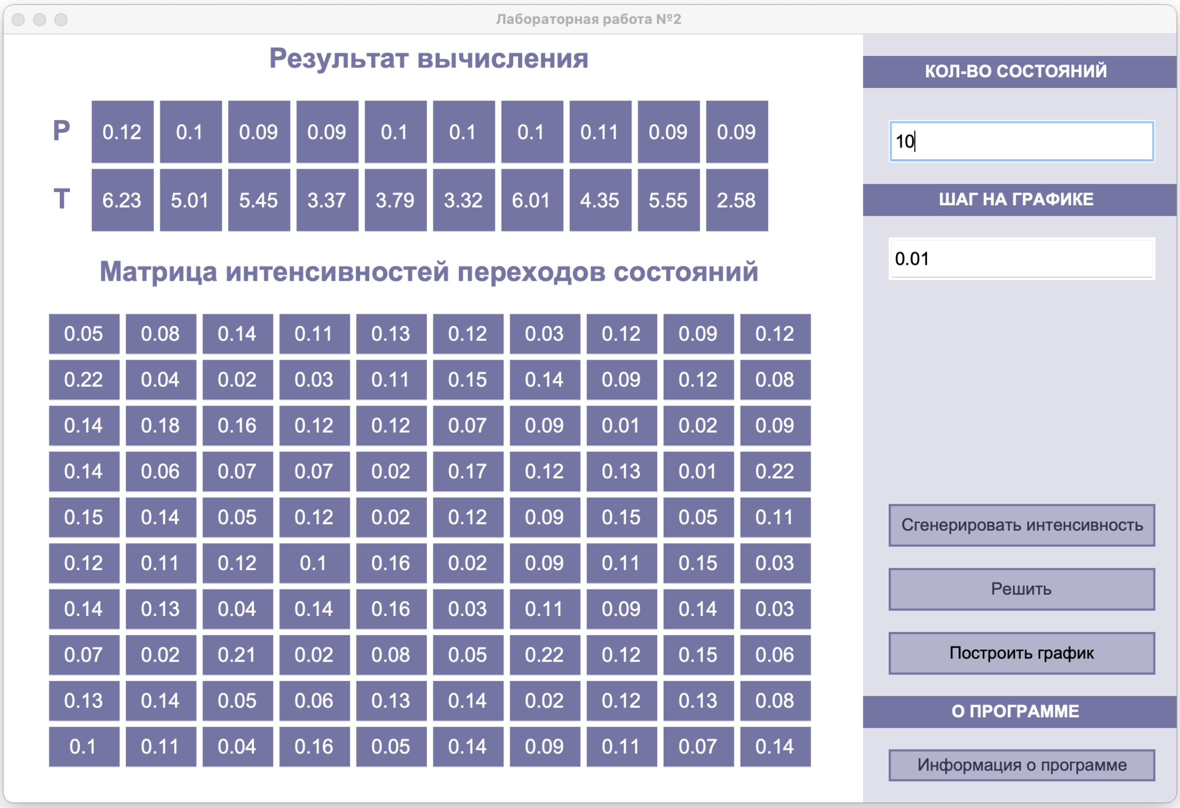


Рисунок 3.3 – Система из 10 состояний

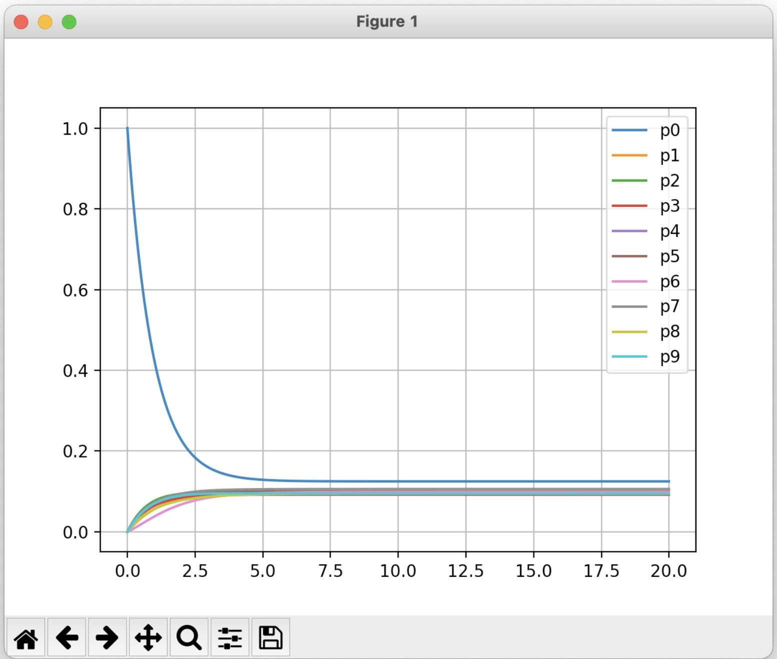


Рисунок 3.4 – График вероятности от времени для системы из 10 состояний