|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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»

**Лабораторная работа №2**

**по дисциплине «Моделирование»**

|  |  |
| --- | --- |
| **Тема:** Цепи Маркова  **Группа:** ИУ7-73Б  **Студент:** Сукочева Алис  **Преподаватель:** Рудаков И. В. |  |

Москва.

2021 г.

**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.

# 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.

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).

## Пример

Система имеет 3 состояния с матрицей интенсивностей, описанной в табл. 1.

Таблица 1 – матрица интенсивностей

|  |  |  |
| --- | --- | --- |
| 0 |  |  |
|  | 0 |  |
|  |  | 0 |

(1)

Для получения предельных вероятностей, то есть вероятностей в стационарном режиме работы при необходимо приравнять левые части уравнений к нулю. Таким образом получается система линейных уравнений. Для решения полученной системы необходимо добавить условие нормировки ().

Также необходимо найти время, в которое достигается вероятностная константа. В общем случае для решения данной задачи необходимо решить систему ОДУ (1) в общем виде. Даже библиотечные функции (в ЯП Python) не находят функции как таковые, а находят только значения для функций из системы ОДУ в заданных точках, поэтому для решения данной задачи можно воспользоваться численным методом: найдем все значения вероятности , как функции времени, с интервалом в некотором интервале [t0, tN]. Когда найденная вероятность будет равна найденной (при итерации с конца!) ранее вероятностной константе с точностью до заданной погрешности, тогда можно считать искомое время найденным. На каждом шаге необходимо вычислять приращения для каждой вероятности:

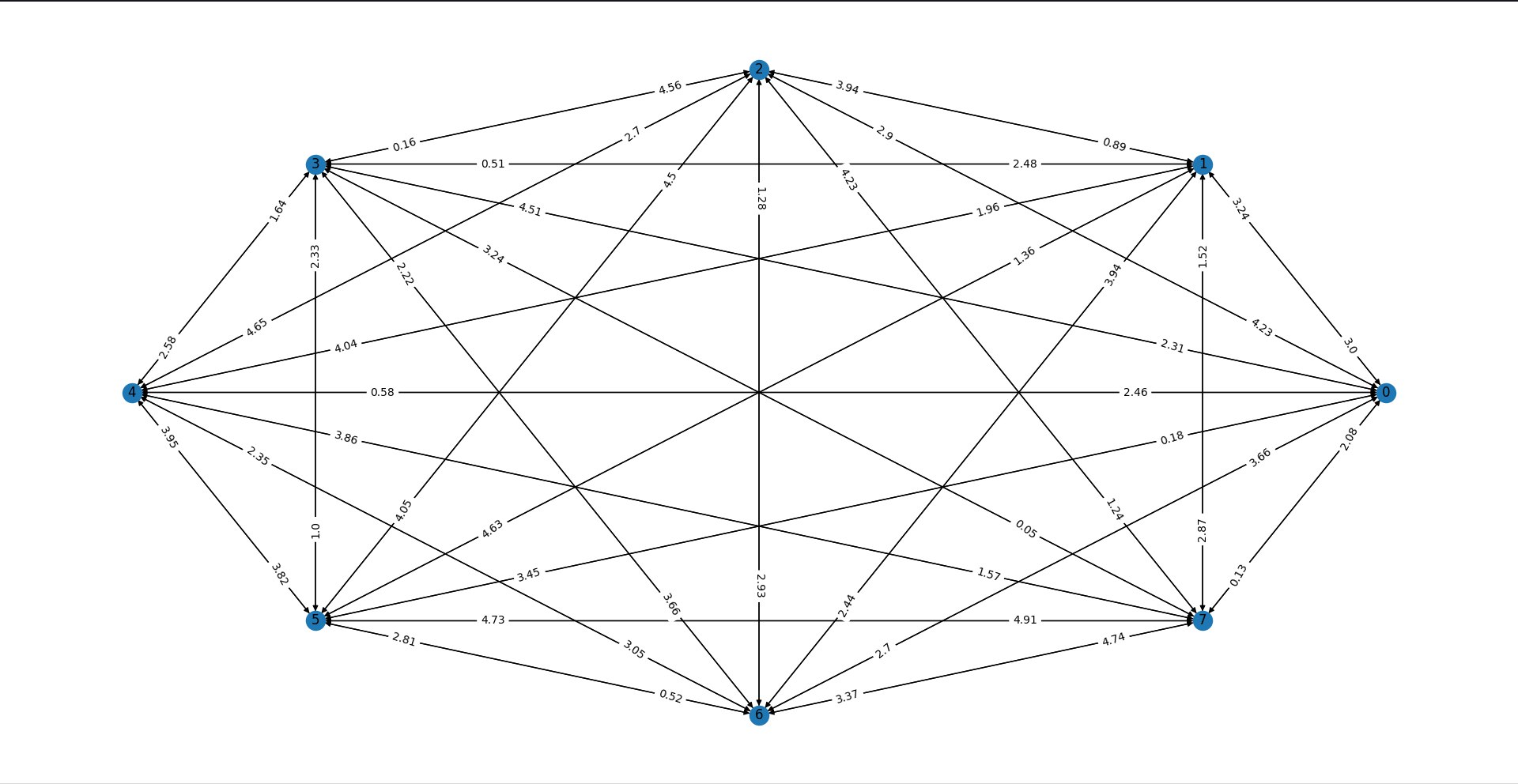
.

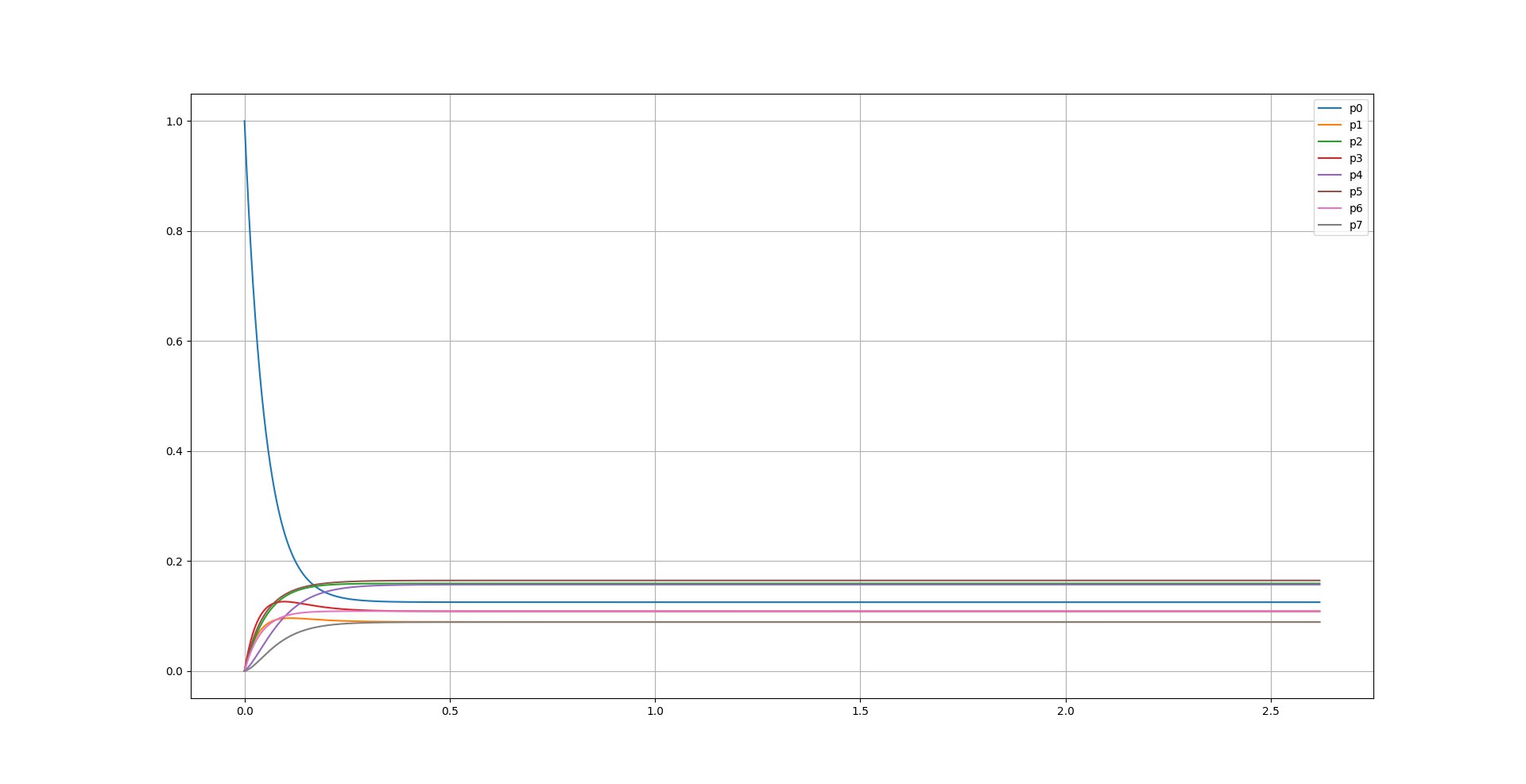
Начальные значения для разумно брать на основе интенсивностей в системе, будет рассмотрено далее.

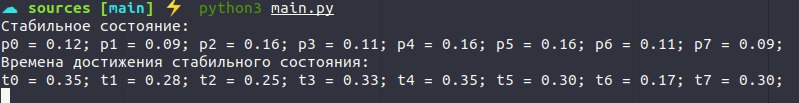
**Пример работы**

**1. Общий случай**

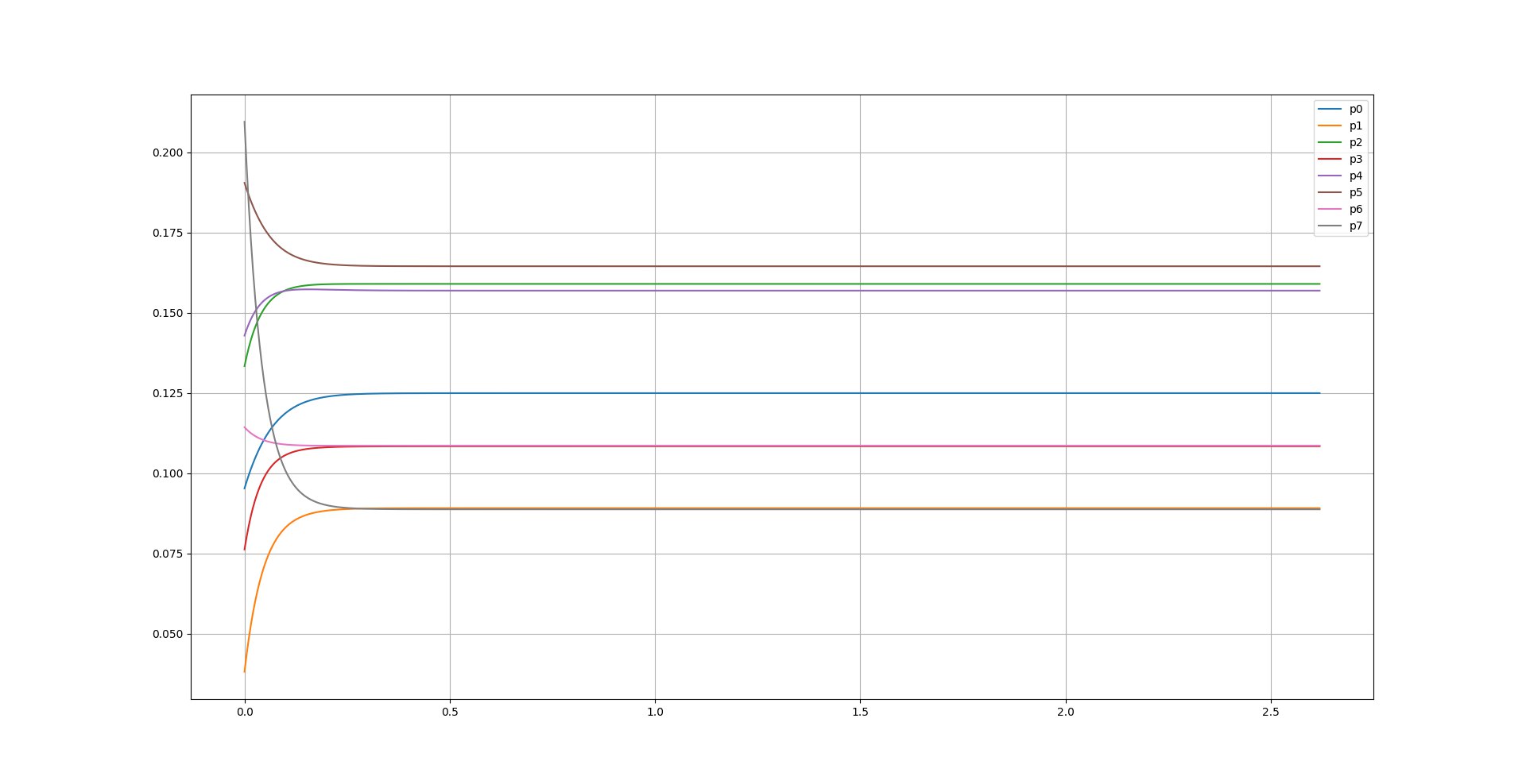
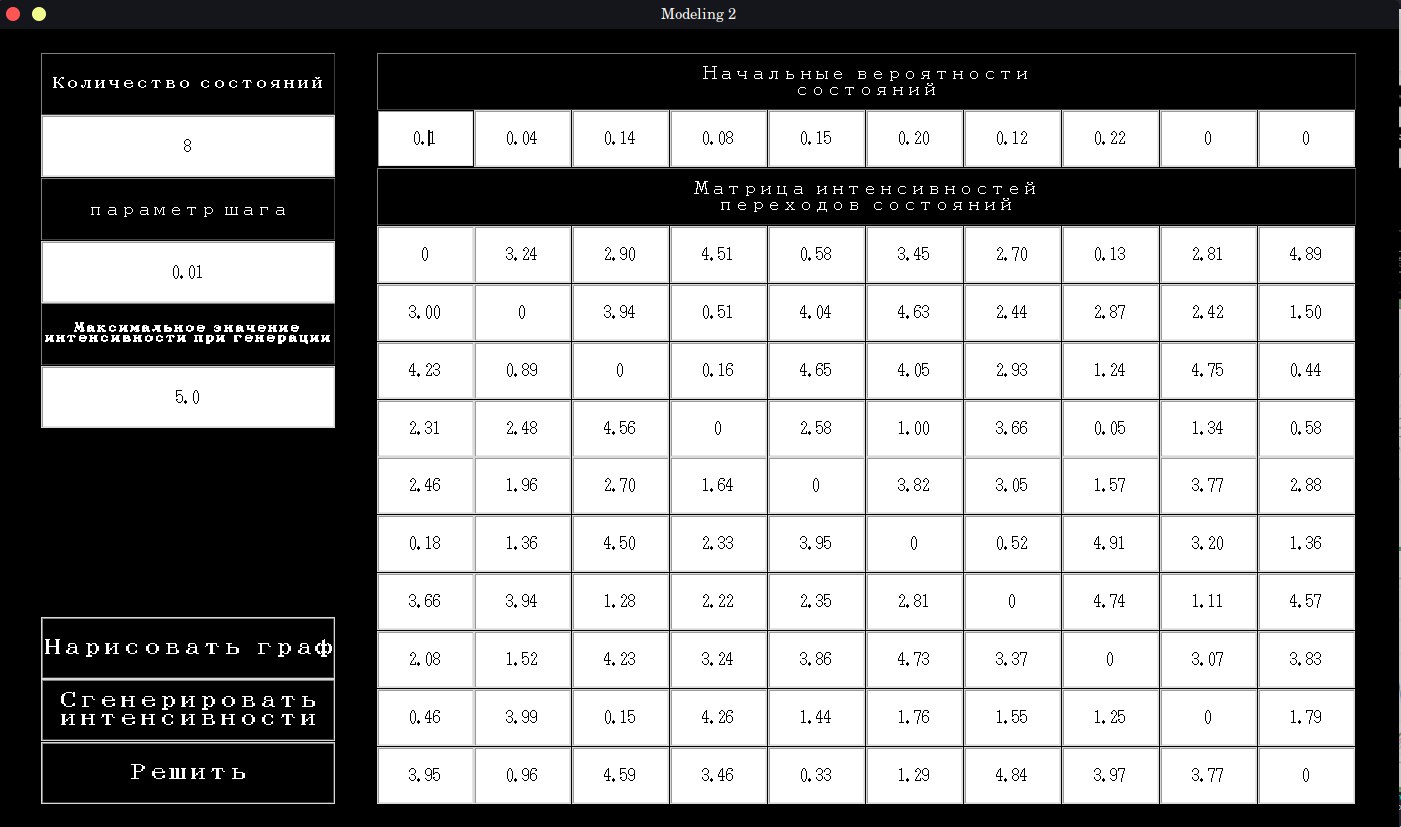
****

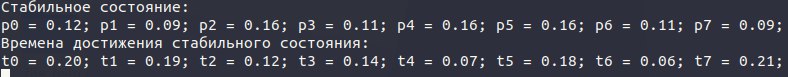
****



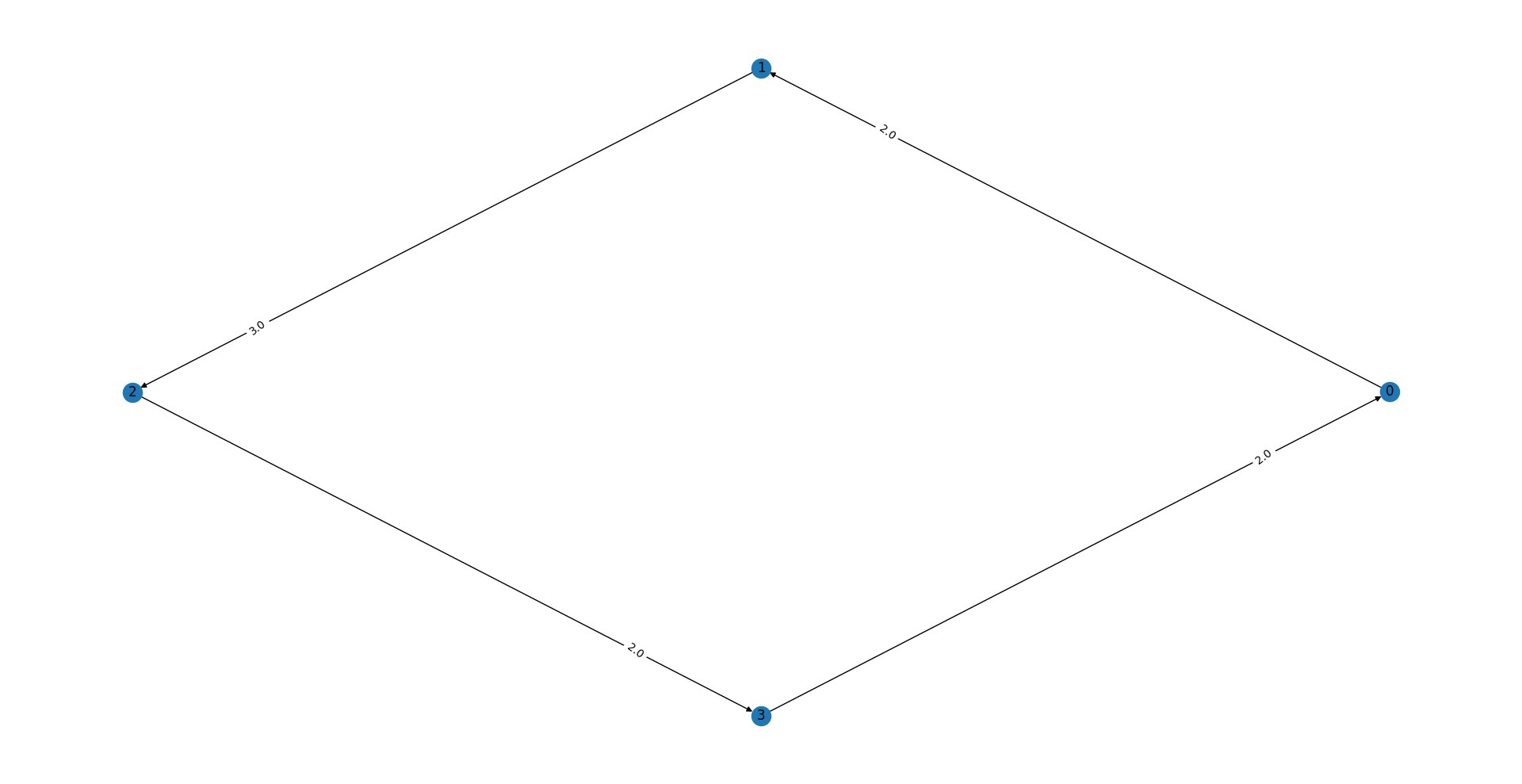
****

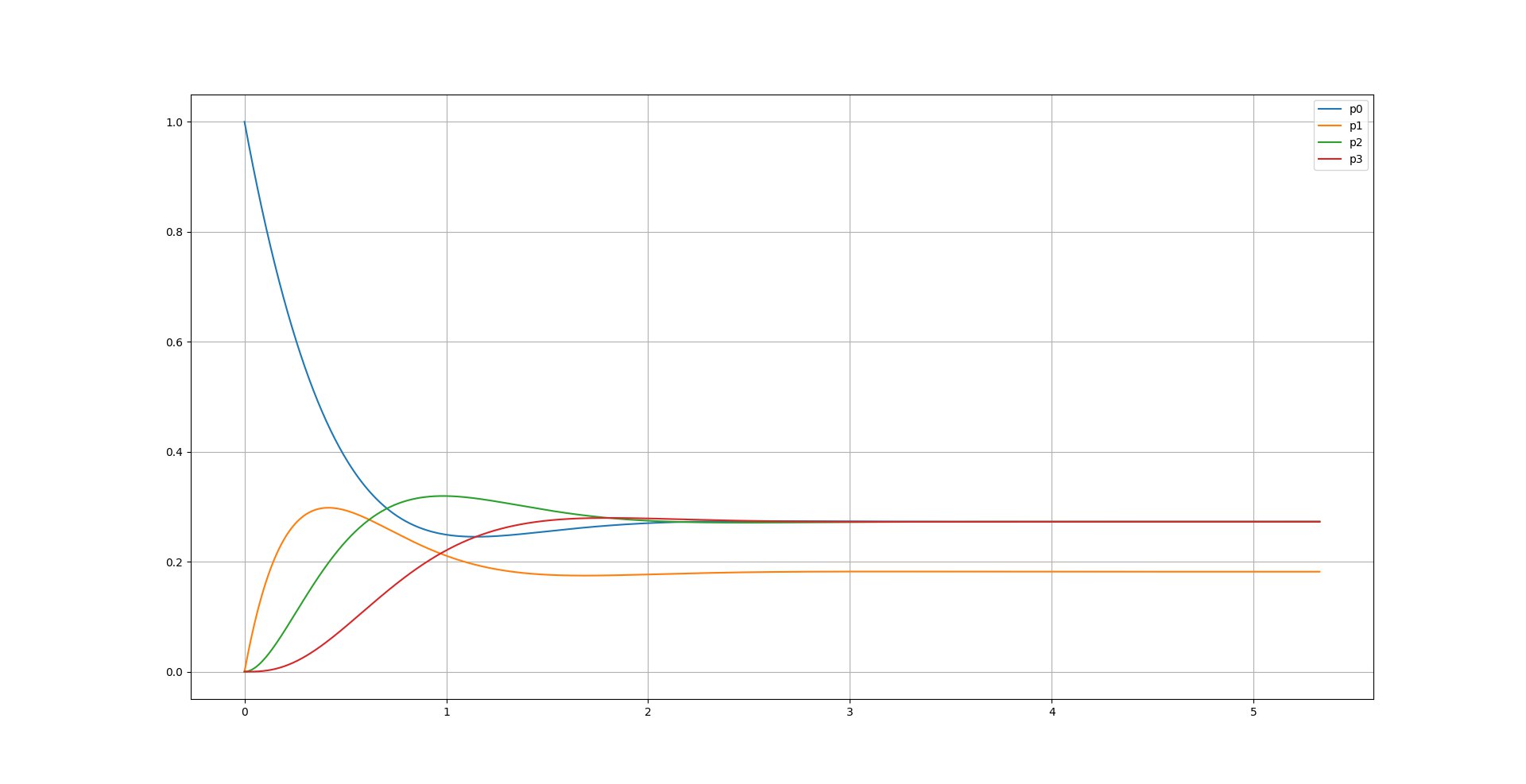
**2. Общий случай с неравномерным случайным начальным состоянием**

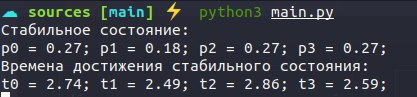
****

****

**3. Кольцо.**

****

****

****

**Текст программы**

## ****main.py**** :

|  |  |
| --- | --- |
|  | import random  import math |
|  | import tkinter as tk |
|  | import config as cfg |
|  | import tkinter.messagebox as mb |
|  | import matplotlib.pyplot as plt |
|  | import numpy as np |
|  | from algorithm import solve |
|  | import networkx as nx |
|  |  |
|  | root = tk.Tk() |
|  | root.title("Modeling 2") |
|  | root["bg"] = cfg.MAIN\_COLOUR |
|  | root.geometry(str(cfg.WINDOW\_WIDTH) + "x" + str(cfg.WINDOW\_HEIGHT)) |
|  | root.resizable(height=False, width=False) |
|  |  |
|  | data\_frame = tk.Frame(root) |
|  | data\_frame["bg"] = cfg.MAIN\_COLOUR |
|  |  |
|  | matrix\_frame = tk.Frame(root) |
|  | matrix\_frame["bg"] = cfg.MAIN\_COLOUR |
|  |  |
|  | data\_frame.place(x=int(cfg.BORDERS\_WIDTH), y=int(cfg.BORDERS\_HEIGHT), |
|  | width=cfg.DATA\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT) |
|  |  |
|  | matrix\_frame.place(x=int(cfg.BORDERS\_WIDTH \* 2 + cfg.DATA\_WIDTH), y=int(cfg.BORDERS\_HEIGHT), |
|  | width=cfg.MATRIX\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT) |
|  |  |
|  | def process(): |
|  | global matrix\_entries, dt\_entry, matrix\_size\_entry, start\_probs\_entries |
|  | size = int(matrix\_size\_entry.get()) |
|  | matrix = [[float(matrix\_entries[i][j].get()) for j in range(size)] for i in range(size)] |
|  | start\_probs = [float(start\_probs\_entries[i].get()) for i in range(size)] |
|  | dt = float(dt\_entry.get()) |
|  | solve(matrix, start\_probs, dt) |
|  |  |
|  |  |
|  | def draw\_graph(): |
|  | global matrix\_entries, matrix\_size\_entry |
|  | size = int(matrix\_size\_entry.get()) |
|  | matrix = [[float(matrix\_entries[i][j].get()) for j in range(size)] for i in range(size)] |
|  |  |
|  | G = nx.from\_numpy\_matrix(np.matrix(matrix), create\_using=nx.DiGraph) |
|  | layout = nx.circular\_layout(G) |
|  | nx.draw(G, layout, with\_labels=True) |
|  | nx.draw\_networkx\_edge\_labels(G, pos=layout, edge\_labels=nx.get\_edge\_attributes(G,'weight'), label\_pos=0.2) |
|  | plt.show() |
|  |  |
|  |  |
|  | start\_probs\_entries = [ |
|  | tk.Entry(matrix\_frame, bg=cfg.ADD\_COLOUR, font=("Arial", 12), fg=cfg.MAIN\_COLOUR, justify="center") |
|  | for i in range(cfg.MATRIX\_MAX\_SIZE) |
|  | ] |
|  |  |
|  | for i in range(1, cfg.MATRIX\_MAX\_SIZE): |
|  | start\_probs\_entries[i].insert(0, '0') |
|  | start\_probs\_entries[0].insert(0, '1') |
|  |  |
|  | matrix\_entries = [ |
|  | [ |
|  | tk.Entry(matrix\_frame, bg=cfg.ADD\_COLOUR, font=("Arial", 12), fg=cfg.MAIN\_COLOUR, justify="center") |
|  | for i in range(cfg.MATRIX\_MAX\_SIZE) |
|  | ] |
|  | for j in range(cfg.MATRIX\_MAX\_SIZE) |
|  | ] |
|  |  |
|  | for i in range(cfg.MATRIX\_MAX\_SIZE): |
|  | for j in range(cfg.MATRIX\_MAX\_SIZE): |
|  | matrix\_entries[i][j].insert(0, '0') |
|  |  |
|  | def generate\_random(): |
|  | global lambda\_limit\_entry, matrix\_entries |
|  | limit = float(lambda\_limit\_entry.get()) |
|  | for i in range(cfg.MATRIX\_MAX\_SIZE): |
|  | for j in range(cfg.MATRIX\_MAX\_SIZE): |
|  | matrix\_entries[i][j].delete(0, len(matrix\_entries[i][j].get())) |
|  | matrix\_entries[i][j].insert(0, '0' if i == j else f"{random.random() \* limit:.2f}") |
|  |  |
|  | matrix\_size\_entry = tk.Entry(data\_frame, bg=cfg.ADD\_COLOUR, font=("Arial", 12), |
|  | fg=cfg.MAIN\_COLOUR, justify="center") |
|  | matrix\_size\_entry.insert(0, '2') |
|  |  |
|  | dt\_entry = tk.Entry(data\_frame, bg=cfg.ADD\_COLOUR, font=("Arial", 12), |
|  | fg=cfg.MAIN\_COLOUR, justify="center") |
|  | dt\_entry.insert(0, '0.1') |
|  |  |
|  | lambda\_limit\_entry = tk.Entry(data\_frame, bg=cfg.ADD\_COLOUR, font=("Arial", 12), |
|  | fg=cfg.MAIN\_COLOUR, justify="center") |
|  | lambda\_limit\_entry.insert(0, '1.0') |
|  |  |
|  | lambda\_limit\_label = tk.Label(data\_frame, text="Максимальное значение\nинтенсивности при генерации", |
|  | font=("Arial", 8), bg=cfg.MAIN\_COLOUR, |
|  | fg=cfg.ADD\_COLOUR, relief=tk.GROOVE) |
|  |  |
|  | start\_probs\_label = tk.Label(matrix\_frame, text="Начальные вероятности\nсостояний", |
|  | font=("Arial", 12), bg=cfg.MAIN\_COLOUR, |
|  | fg=cfg.ADD\_COLOUR, relief=tk.GROOVE) |
|  |  |
|  | matrix\_label = tk.Label(matrix\_frame, text="Матрица интенсивностей\nпереходов состояний", font=("Arial", 12), |
|  | bg=cfg.MAIN\_COLOUR, fg=cfg.ADD\_COLOUR, relief=tk.GROOVE) |
|  |  |
|  | matrix\_size\_label = tk.Label(data\_frame, text="Количество состояний", font=("Arial", 10), |
|  | bg=cfg.MAIN\_COLOUR, fg=cfg.ADD\_COLOUR, relief=tk.GROOVE) |
|  |  |
|  | dt\_label = tk.Label(data\_frame, text="параметр шага", font=("Arial", 12), |
|  | bg=cfg.MAIN\_COLOUR, fg=cfg.ADD\_COLOUR, relief=tk.GROOVE) |
|  |  |
|  | lambda\_gen\_button = tk.Button(data\_frame, text="Сгенерировать\nинтенсивности", font=("Consolas", 14), |
|  | bg=cfg.MAIN\_COLOUR, fg=cfg.ADD\_COLOUR, command=generate\_random, |
|  | activebackground=cfg.ADD\_COLOUR, activeforeground=cfg.MAIN\_COLOUR) |
|  | draw\_graph\_button = tk.Button(data\_frame, text="Нарисовать граф", font=("Consolas", 14), |
|  | bg=cfg.MAIN\_COLOUR, fg=cfg.ADD\_COLOUR, command=draw\_graph, |
|  | activebackground=cfg.ADD\_COLOUR, activeforeground=cfg.MAIN\_COLOUR) |
|  | solve\_button = tk.Button(data\_frame, text="Решить", font=("Consolas", 14), |
|  | bg=cfg.MAIN\_COLOUR, fg=cfg.ADD\_COLOUR, command=process, |
|  | activebackground=cfg.ADD\_COLOUR, activeforeground=cfg.MAIN\_COLOUR) |
|  |  |
|  |  |
|  |  |
|  | offset = 0 |
|  | matrix\_size\_label.place(x=0, y=cfg.DATA\_HEIGHT \* offset // cfg.ROWS, width=cfg.DATA\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT // cfg.ROWS) |
|  | offset += 1 |
|  | matrix\_size\_entry.place(x=0, y=cfg.DATA\_HEIGHT \* offset // cfg.ROWS, width=cfg.DATA\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT // cfg.ROWS) |
|  | offset += 1 |
|  | dt\_label.place(x=0, y=cfg.DATA\_HEIGHT \* offset // cfg.ROWS, width=cfg.DATA\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT // cfg.ROWS) |
|  | offset += 1 |
|  | dt\_entry.place(x=0, y=cfg.DATA\_HEIGHT \* offset // cfg.ROWS, width=cfg.DATA\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT // cfg.ROWS) |
|  | offset += 1 |
|  | lambda\_limit\_label.place(x=0, y=cfg.DATA\_HEIGHT \* offset // cfg.ROWS, width=cfg.DATA\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT // cfg.ROWS) |
|  | offset += 1 |
|  | lambda\_limit\_entry.place(x=0, y=cfg.DATA\_HEIGHT \* offset // cfg.ROWS, width=cfg.DATA\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT // cfg.ROWS) |
|  |  |
|  | offset = cfg.ROWS - 3 |
|  | draw\_graph\_button.place(x=0, y=cfg.DATA\_HEIGHT \* offset // cfg.ROWS, width=cfg.DATA\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT // cfg.ROWS) |
|  | offset += 1 |
|  | lambda\_gen\_button.place(x=0, y=cfg.DATA\_HEIGHT \* offset // cfg.ROWS, width=cfg.DATA\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT // cfg.ROWS) |
|  | offset += 1 |
|  | solve\_button.place(x=0, y=cfg.DATA\_HEIGHT \* offset // cfg.ROWS, width=cfg.DATA\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT // cfg.ROWS) |
|  |  |
|  | offset = 0 |
|  |  |
|  | start\_probs\_label.place(x=0, y=cfg.DATA\_HEIGHT \* offset // cfg.MATRIX\_FRAME\_ROWS, width=cfg.MATRIX\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT // cfg.MATRIX\_FRAME\_ROWS) |
|  |  |
|  | offset += 1 |
|  |  |
|  | for i in range(cfg.MATRIX\_MAX\_SIZE): |
|  | start\_probs\_entries[i].place(x=int(i / cfg.MATRIX\_MAX\_SIZE \* cfg.MATRIX\_WIDTH), y=cfg.DATA\_HEIGHT \* offset // cfg.MATRIX\_FRAME\_ROWS, |
|  | width=cfg.MATRIX\_WIDTH // cfg.MATRIX\_MAX\_SIZE, height=cfg.DATA\_HEIGHT // cfg.MATRIX\_FRAME\_ROWS) |
|  |  |
|  | offset += 1 |
|  |  |
|  | matrix\_label.place(x=0, y=cfg.DATA\_HEIGHT \* offset // cfg.MATRIX\_FRAME\_ROWS, width=cfg.MATRIX\_WIDTH, |
|  | height=cfg.DATA\_HEIGHT // cfg.MATRIX\_FRAME\_ROWS) |
|  |  |
|  | offset += 1 |
|  |  |
|  | for i in range(cfg.MATRIX\_MAX\_SIZE): |
|  | for j in range(cfg.MATRIX\_MAX\_SIZE): |
|  | matrix\_entries[i][j].place(x=int(j / cfg.MATRIX\_MAX\_SIZE \* cfg.MATRIX\_WIDTH), |
|  | y=cfg.DATA\_HEIGHT \* (offset + i) // cfg.MATRIX\_FRAME\_ROWS, |
|  | width=cfg.MATRIX\_WIDTH // cfg.MATRIX\_MAX\_SIZE, |
|  | height=cfg.DATA\_HEIGHT // cfg.MATRIX\_FRAME\_ROWS) |
|  | root.mainloop() |

## ****algorithm.py**** :

|  |
| --- |
|  |
|  | import numpy as np  from scipy.integrate import odeint |
|  | import matplotlib.pyplot as plt |
|  |  |
|  | eps = 1e-3 |
|  |  |
|  | def normalize(probs): |
|  | s = sum(probs) |
|  | for i in range(len(probs)): |
|  | probs[i] /= s |
|  |  |
|  |  |
|  | def solve\_ode(matrix, start\_probs, dt, steady\_states): |
|  | matrix\_to\_solve = [ |
|  | [-sum(matrix[i]) if j == i else matrix[j][i] for j in range(len(matrix))] |
|  | for i in range(len(matrix))] |
|  | ts = np.arange(0, dt \* 1000, dt) |
|  | results = odeint(vectorfields, start\_probs, ts, |
|  | args=(matrix\_to\_solve,), atol=1.0e-8, rtol=1.0e-6) |
|  | results = np.transpose(results) |
|  | steady\_ts = [] |
|  |  |
|  | for i in range(len(results)): |
|  | plt.plot(ts, results[i], label='p' + str(i)) |
|  | 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) |
|  |  |
|  | print("Времена достижения стабильного состояния:") |
|  | for i in range(len(steady\_ts)): |
|  | print(f"t{str(i)} = {steady\_ts[i]:.2f}; ", end='') |
|  | print() |
|  |  |
|  | plt.legend() |
|  | plt.grid() |
|  | plt.show() |
|  |  |
|  | def vectorfields(w, \_, matrix\_to\_solve): |
|  | """ |
|  | Defines the differential equations for the coupled spring-mass system. |
|  |  |
|  | Arguments: |
|  | w : vector of the state variables: |
|  | w = [p1, p2, p3...] |
|  | \_ : time |
|  | matrix\_to\_solve : vector of the parameters: |
|  | lambdas = [[lambda\_ji for i in range(len(matrix))] for j in range(len(matrix))] |
|  | """ |
|  | 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 solve(matrix, start\_probs, dt): |
|  | normalize(start\_probs) |
|  | b = [0 for \_ in range(len(matrix) - 1)] |
|  | b.append(1) |
|  | matrix\_to\_solve = [ |
|  | [-sum(matrix[i]) if j == i else matrix[j][i] for j in range(len(matrix))] |
|  | for i in range(len(matrix) - 1)] |
|  | matrix\_to\_solve.append([1 for \_ in range(len(matrix))]) |
|  |  |
|  | ps = np.linalg.solve(matrix\_to\_solve, b) |
|  | print("Стабильное состояние:") |
|  | for i in range(len(ps)): |
|  | print(f"p{str(i)} = {ps[i]:.2f}; ", end='') |
|  | print() |
|  | max\_lambda = max([max(matrix[i]) for i in range(len(matrix))]) |
|  | avg\_lambda = sum([sum(matrix[i]) for i in range(len(matrix))]) / len(matrix) / (len(matrix) - 1) |
|  | dt = (1 / (max\_lambda + avg\_lambda) \* 2) \* dt |
|  | solve\_ode(matrix, start\_probs, dt, ps) |