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 дисциплине
"Моделирование"

Тема Марковские процессы

Студент Малышев И. А.

Группа ИУ7-71Б

Оценка (баллы) _____

Преподаватель: Рудаков И. В.

Москва — 2022 г.

1 | Задание

Реализовать программу, которая позволяет определить время пребывания в каждом состоянии в установившемся режиме работы системы массового обслуживания.

2 | Решение

2.1 Теоретическая часть

Случайный процесс, протекающий в некоторой системе, называют марковским, если он обладает следующим свойством: для каждого момента времени t_0 вероятность любого состояния системы в будущем (t $> t_0$) зависит только от её состояния в настоящем и не зависит от того, когда и каким образом система пришла в это состояние (как процесс развивался в прошлом).

Функционирование системы может быть задано размеченным графом, где дуги обозначают интенсивности переходов, а узлы – состояния системы.

Для решения поставленной задачи может быть составлена система, состоящая из уравнений Колмогорова, каждое из которых имеет вид:

$$\frac{dp_i(t)}{dt} = \sum_{j=1}^{n} \lambda_{ji} p_j(t) - p_i(t) \sum_{j=1}^{n} \lambda_{ij} \tag{2.1}$$

где $p_i(t)$ – вероятность нахождения системы в состоянии S_i в момент времени t, n – количество состояний в системе, $\lambda_i j$ – интенсивность перехода системы из состояния S_i в состояние S_j. Для определения предельных вероятностей в построенной системе уравнений Колмогорова производные приравниваются нулю и одно из уравнений заменяется на уравнение нормировки для установившегося режима работы системы:

$$\sum_{j=1}^{n} p_j(t) = 1 \tag{2.2}$$

Для определения точки стабилизации системы можно определять вероятности нахождения в определённых состояниях с некоторым малым шагом Δt. Точка стабилизации будет определена в случае, когда будет выполнено условие того, что приращение вероятности после шага, как и разница между предельной вероятностью состояния и вычисленной вероятностью, достаточно мала: $|p_j(t + \Delta t) - p_j(t)| < \epsilon$ и $|p_j(t) - \lim_{t\to\infty} p_j(t)| < \epsilon$ где ϵ может, например, принять значение 10^{-3}.

2.2 Листинг

Далее представлен фрагмент программы, выполняющий поставленнуое задание.

```
1 internal static class KolmogorovMath
2 {
3   public const double TimeDelta = 1e-3;
4   public const int MaxStatesCount = 10;
5
6   public static Matrix<double> GetUltimatePropabilities(Matrix<double> matrix)
7   {
```

```
8       var coefsMatrix = BuildCoefsMatrix(matrix);
9       var augmMatrix = BuildAugmentationMatrix(matrix.RowCount);
10
11      return coefsMatrix.Solve(augmMatrix);
12    }
13
14    public static IEnumerable<double> GetStabilizationTimes(Matrix<double> matrix,
          Vector<double> startPropabilities, Vector<double> ultimatePropabilities)
15    {
16      int n = matrix.RowCount;
17      double currTime = 0;
18      Vector<double> currPropabilities = startPropabilities.Clone();
19      Vector<double> stabilizationTimes = Vector<double>.Build.Dense(n);
20
21      double totalLambdaSum = matrix.RowSums().Sum() * MaxStatesCount;
22      double[] Eps = ultimatePropabilities.Select(p => p /
            totalLambdaSum).ToArray();
23
24      while (!stabilizationTimes.All(p => Math.Abs(p) >= 1e-3))
25      {
26        var currDps = Dps(matrix, currPropabilities).ToArray();
27        for (int i = 0; i < n; i++)
28        {
29          if (Math.Abs(stabilizationTimes[i]) < 1e-3 && currDps[i] <= Eps[i] &&
                Math.Abs(currPropabilities[i] - ultimatePropabilities[i]) <= Eps[i])
30          stabilizationTimes[i] = currTime;
31
32          currPropabilities[i] += currDps[i];
33        }
34
35        currTime += TimeDelta;
36      }
37
38      return stabilizationTimes;
39    }
40
41    public static IEnumerable<IEnumerable<PointF>>
          PropabilityOverTime(Matrix<double> matrix, Vector<double>
          startPropabilities, double endTime)
42    {
43      int n = matrix.RowCount;
44      double currTime = 0;
45      Vector<double> currPropabilities = startPropabilities.Clone();
46
47      List<PointF>[] listOfPoints = new List<PointF>[startPropabilities.Count];
48
49      for (int i = 0; i < startPropabilities.Count; i++)
50      listOfPoints[i] = new List<PointF>();
51
52      while (currTime < endTime)
53      {
54        for (int i = 0; i < startPropabilities.Count; i++)
```

```
55        listOfPoints[i].Add(new PointF((float)currTime,
              (float)currPropabilities[i]));
56
57        var currDps = Dps(matrix, currPropabilities);
58        for (int i = 0; i < currPropabilities.Count; i++)
59        currPropabilities[i] += currDps.ElementAt(i);
60
61        currTime += TimeDelta;
62      }
63
64      return listOfPoints;
65    }
66
67    static IEnumerable<double> Dps(Matrix<double> matrix, Vector<double>
          probabilities)
68    {
69      int n = matrix.RowCount;
70
71      double[] res = new double[n];
72
73      for (int i = 0; i < n; i++)
74      {
75        double sum = 0;
76
77        for (int j = 0; j < n; j++)
78        sum += i != j ? probabilities[j] * matrix[j, i] : probabilities[j] *
              (matrix[i, i] - matrix.RowSums()[i]);
79
80        res[i] = sum * TimeDelta;
81      }
82
83      return res;
84    }
85
86    static Matrix<double> BuildAugmentationMatrix(int count)
87    {
88      Matrix<double> matrix = Matrix<double>.Build.Dense(count, 1);
89      matrix[count - 1, 0] = 1;
90
91      return matrix;
92    }
93
94    static Matrix<double> BuildCoefsMatrix(Matrix<double> matrix)
95    {
96      Matrix<double> res = Matrix<double>.Build.Dense(matrix.RowCount,
            matrix.ColumnCount);
97      int n = matrix.RowCount;
98
99      for (int i = 0; i < n - 1; i++)
100     {
101       for (int j = 0; j < n; j++)
102       res[i, i] -= matrix[i, j];
```

```
103         for (int j = 0; j < n; j++)
104         res[i, j] += matrix[j, i];
105       }
106
107       for (int i = 0; i < n; i++)
108       res[n - 1, i] = 1;
109
110       return res;
111     }
112 }
```

2.3 Результаты работы

На рисунках 2.1-2.2 представлен пользовательский интерфейс программы до ввода количества состояний и ввода интенсивностей.

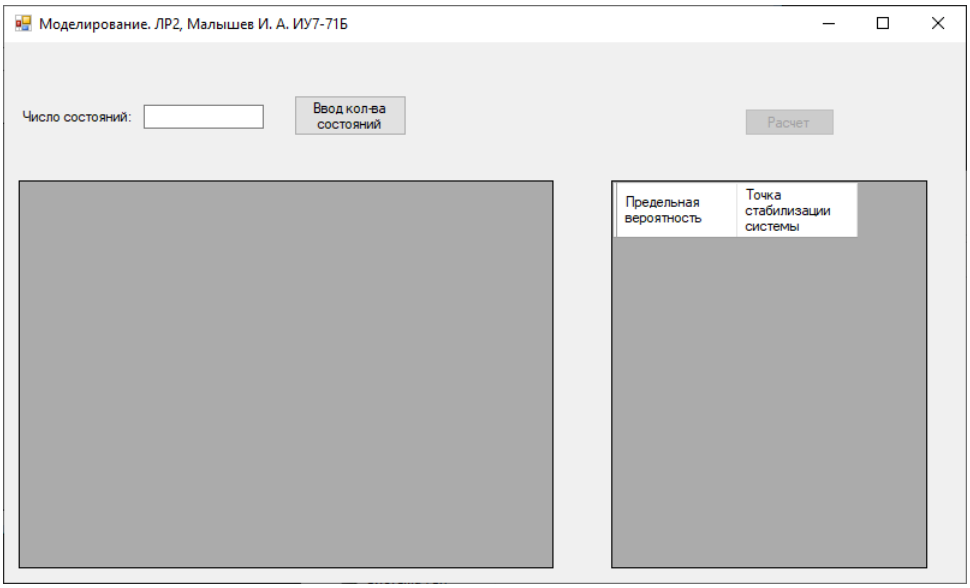

Рис. 2.1: Пользовательский интерфейс программы до ввода количества состояний.

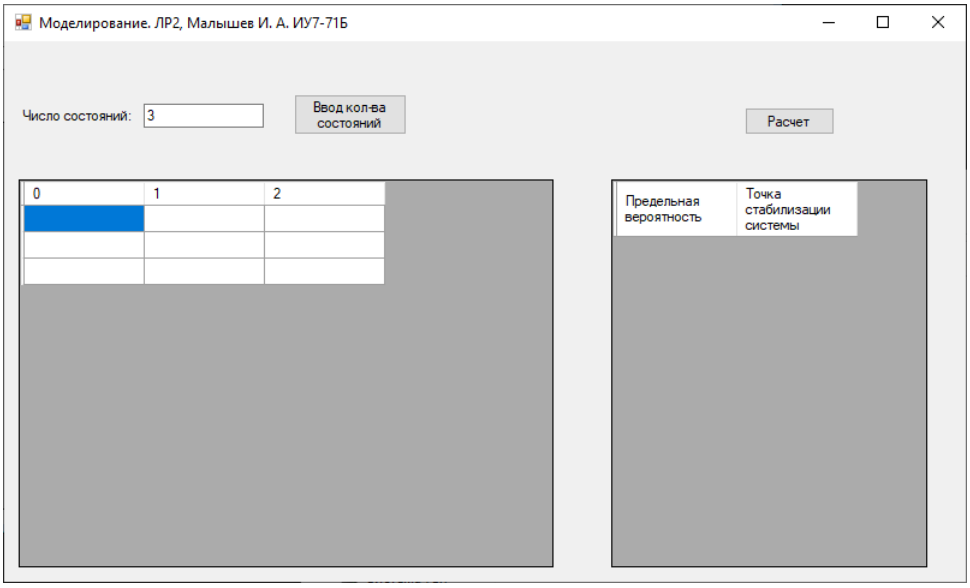

Рис. 2.2: Пользовательский интерфейс программы до ввода интенсивностей.

2.3.1 Пример 1

На рисунках 2.3-2.4 представлен пример результатов работы программы с указанными данными.

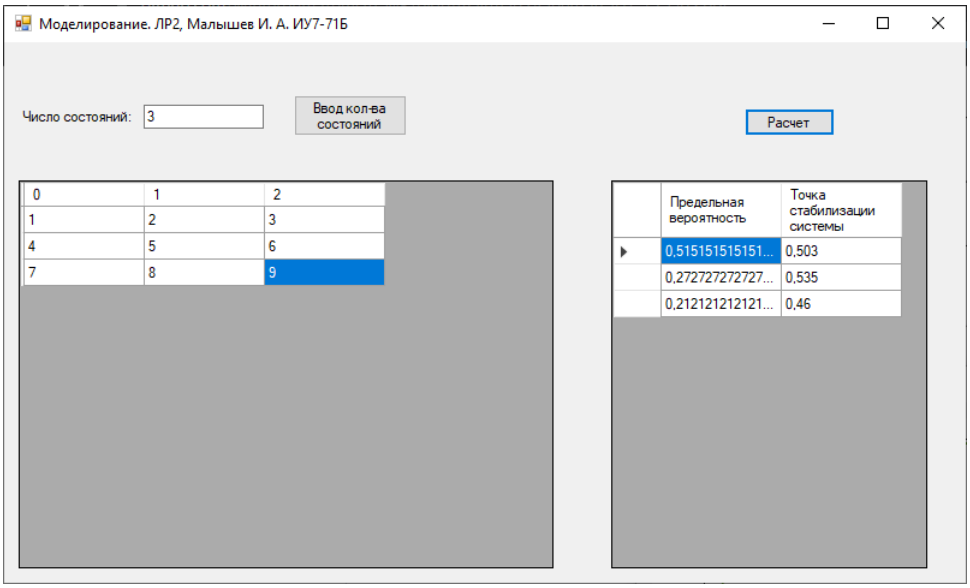

Рис. 2.3: Исходные данные и результат.

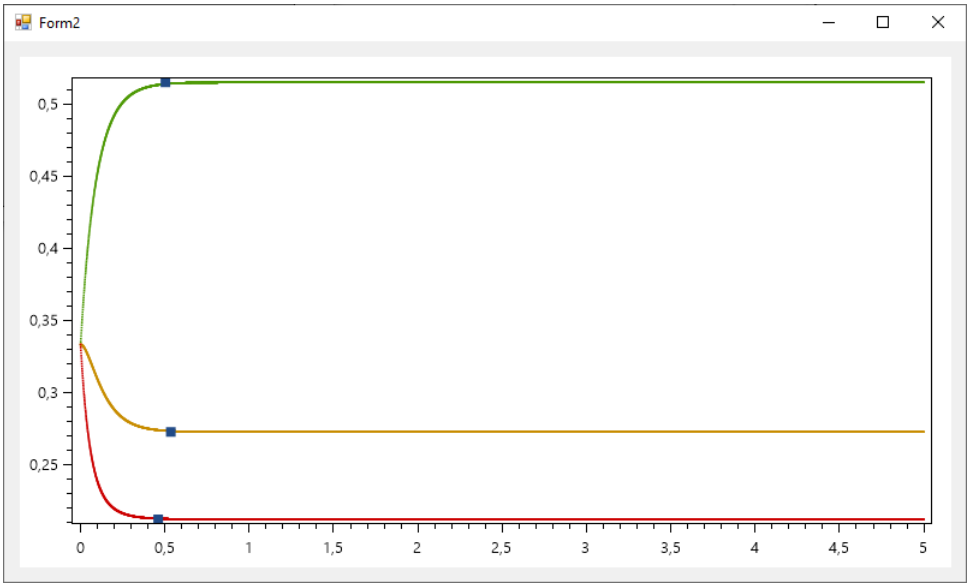

Рис. 2.4: График вероятностей с точками стабилизации.

2.3.2 Пример 2

На рисунках 2.5-2.6 представлен пример результатов работы программы с указанными данными.

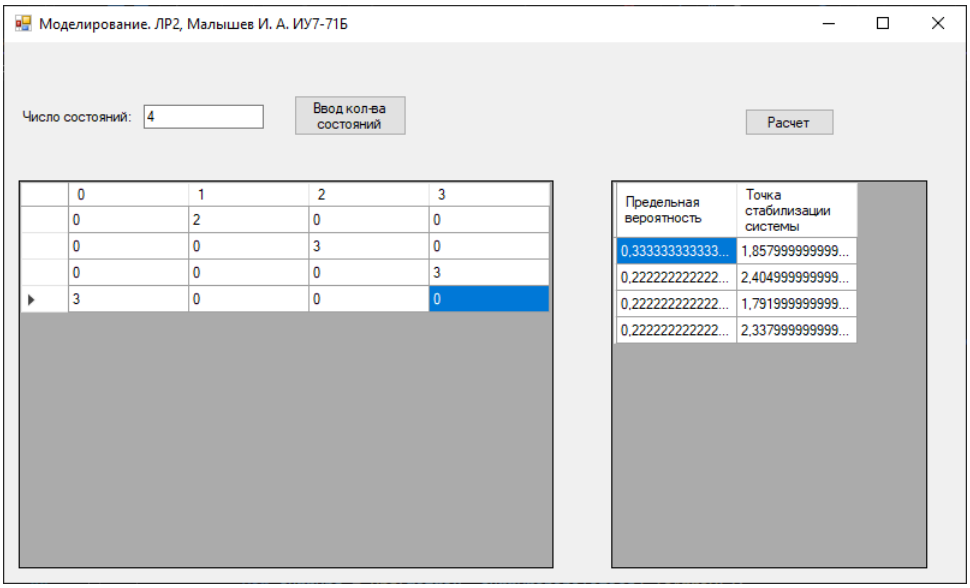

Рис. 2.5: Исходные данные и результат.

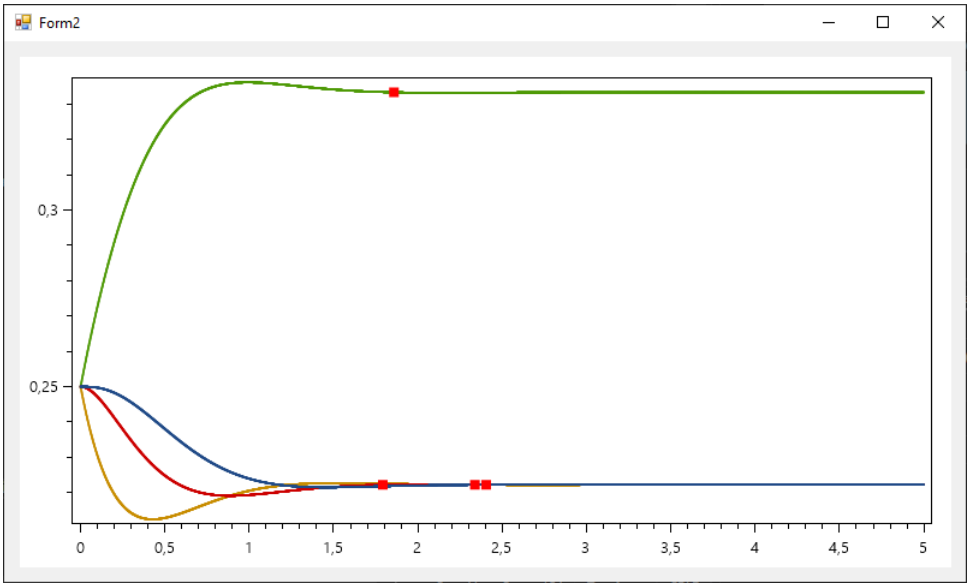

Рис. 2.6: График вероятностей с точками стабилизации.