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 №3

Название Псевдослучайные числа

Дисциплина Моделирование

Студент Зайцева А. А.

Группа ИУ7-72Б

Оценка (баллы) _____

Преподаватель Рудаков И. В.

Москва — 2022 г.

1 Задание

Написать программу, которая генерирует псевдослучайные последовательности одноразрядных, двухразрядных и трехразрядных целых чисел алгоритмическим способом. Также программа может брать готовые псевдослучайные последовательности из файла (табличный способ).

Разработать количественный критерий оценки случайности последовательности чисел. Для каждой сгенерированной или взятой последовательности вычислить и вывести значение критерия. Предусмотреть возможность ввода десяти чисел и оценки их случайности с помощью критерия.

2 Теоретические сведения

На практике наиболее распространены 3 способа получения случайных чисел.

2.1 Аппаратный способ

При использовании аппаратного способа случайные числа вырабатываются специальной электронной приставкой (генератором случайных чисел). Реализация данного способа не требует дополнительных вычислений, необходима только операция – обращение к вычислительному устройству.

В качестве физического эффекта, лежащего в основе генерации случайных чисел, может использоваться, например, шум в электронных приборах. Для генерации необходимы источник шума, ключевая схема, формирователь импульсов и пересчетная схема.

2.2 Табличный способ

В данном способе в качестве источника случайных чисел используют заранее подготовленные таблицы, содержащие проверенные некоррелированные числа. Недостатки такого способа: использование внешнего ресурса для хранения чисел, ограниченность последовательности, предопределенность значений.

2.3 Алгоритмический способ

Алгоритмический генератор является комбинацией физического генератора и детерминированного алгоритма. Такой генератор использует ограниченный набор данных, полученный с выхода физического генератора для создания длинной последовательности чисел преобразованиями исходных чисел. Из-за дороговизны аппаратных генераторов случайных чисел в большинстве случаев, в качестве источника энтропии используются ресурсы вычислительной машины, на которой выполняется программа генерации ПСЧ. При отсутствии аппаратного генератора случайных чисел в качестве источника энтропии могут использоваться:

2.4 Выбранные методы

Для получения случайных чисел алгоритмическим способом выбран линейный конгруэнтный метод.

Линейный конгруэнтный метод

Для осуществления генерации чисел данным методом, необходимо задать 4 числа:

$m > 0$, модуль

Последовательность случайных чисел генерируется при помощи формулы:

(1)

При некоторых наборах чисел m, a, c, и X_0 последовательность не может быть "случайной". Поэтому важно правильно их подобрать. В конгруэнтной последовательности всегда существуют циклы - периоды, необходимо чтобы последовательность, которую мы используем, имела относительно длинный период.

Выбранный критерий оценки случайной последовательности - критерий "хи-квадрат". Это один из самых известных статистических критериев, также это основной метод, используемый в сочетании с другими критериями.

С помощью этого критерия можно узнать, удовлетворяет ли генератор случайных чисел требованию равномерного распределения или нет. Для оценки по этому критерию необходимо вычислить статистику V по формуле:

$$V = \frac{1}{n}\sum_{s=1}^{k}(\frac{Y_s^2}{p_s}) - n \tag{2}$$

где n – количество независимых испытаний, k – количество категорий, Y_s — число наблюдений, которые действительно относятся к категории S, p_s — вероятность того, что каждое наблюдение относится к категории s.

Значение V является значением критерия «хи-квадрат» для экспериментальных данных. Приемлемое значение этого критерия можно определить по таблице 1. Для этого используем строку с v = k-1, где k = 10, 90, 900 для задания лабораторной. P в этой таблице — это вероятность того, что экспериментальное значение Vэксп. будет меньше табулированного (теоретического) Vteop. или равно ему. Ее также можно рассматривать как доверительную вероятность.

Если вычисленное V окажется меньше 1% точки или больше 99%

точки, можно сделать вывод, что эти числа недостаточно случайные. Если V лежит между 1% и 5% точками или между 95% и 99% точками, то эти числа «подозрительны». Если V лежит между 5% и 10% точками или 90%-95% точками, то числа можно считать «почти подозрительными». Проверка по "хи-квадрат"критерию часто производится три раза и более с разными данными. Если по крайней мере два из трех результатов оказываются подозрительными, то числа рассматриваются как недостаточно случайные.

k - 1	p = 1%	p = 5%	p = 25%	p = 50%	p = 75%	p = 95%	p = 99%
9	2.088	3.325	5.899	8.343	11.39	16.92	21.67
89	60.93	68.25	79.68	88.33	97.60	112.02	122.94
899	803.31	830.41	870.05	898.33	927.23	969.86	1000.57

Таблица 1 – Таблица значений Vteop для количества степеней свободы по заданию

3 Результаты работы программы

В начале работы система находится в первом состоянии, ε для определения стабилизации вероятности принимается равным 10^{-5}.

На рисунке 1 приведен пример работы программы для 3 состояний.

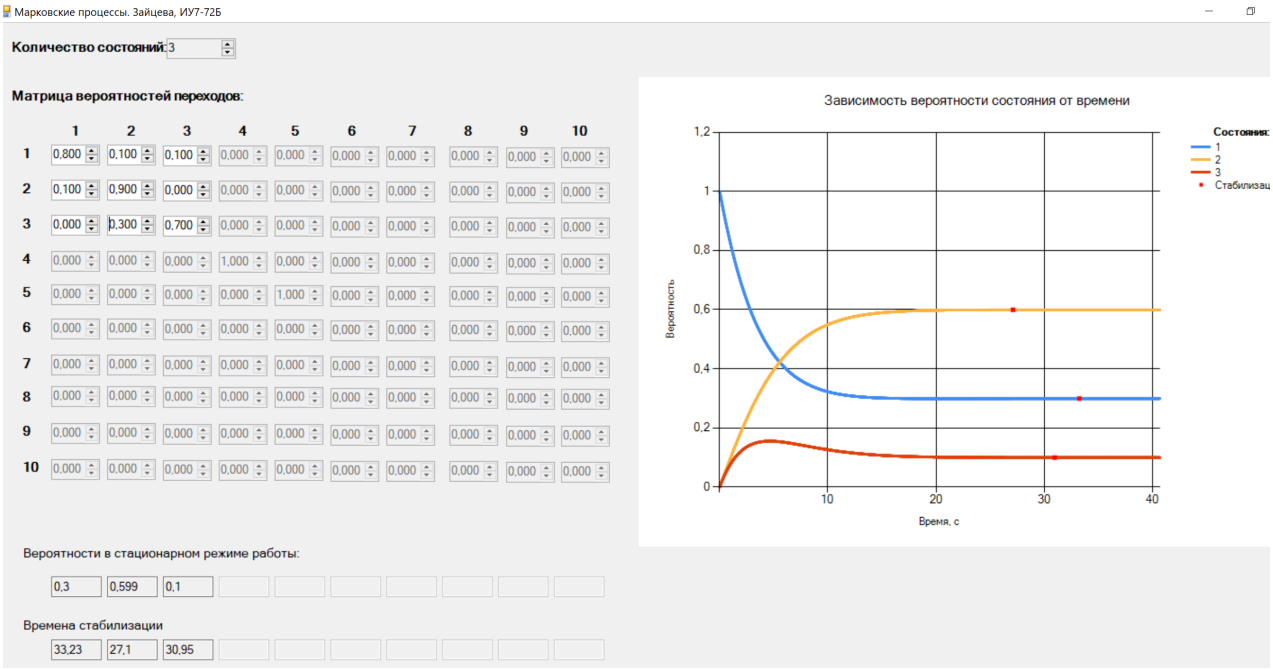

Рисунок 1 – Пример работы программы для 3 состояний

Проверим результаты, приведенные на рисунке выше. Составим систему линейных уравнений для определения вероятностей в стационарном режиме: составим уравнения Колмогорова и приравняем левые части к 0, а также добавим условие нормировки.

$$\begin{cases} 0 = 0.1 \cdot P_2 - 0.1 \cdot P_1 - 0.1 \cdot P_1 \\ 0 = 0.1 \cdot P_1 + 0.3 \cdot P_3 - 0.1 \cdot P2 \\ 0 = 0.1 \cdot P_1 - 0.3 \cdot P_3 \\ P_1 + P_2 + P_3 = 1 \end{cases} \tag{3}$$

$$\begin{cases} P_1 = \frac{3}{10} \\ P_2 = 2 \cdot P_1 = \frac{6}{10} \\ P_3 = \frac{1}{3} \cdot P_1 = \frac{1}{10} \end{cases} \tag{4}$$

Вычисленные значения совпадают с результатами программы.

На рисунке 2 вероятность перехода из второго состояния в любое другое равна 0, поэтому в стабилизировавшемся режиме вероятность этого состояния примерно равна 1, а остальных – 0.

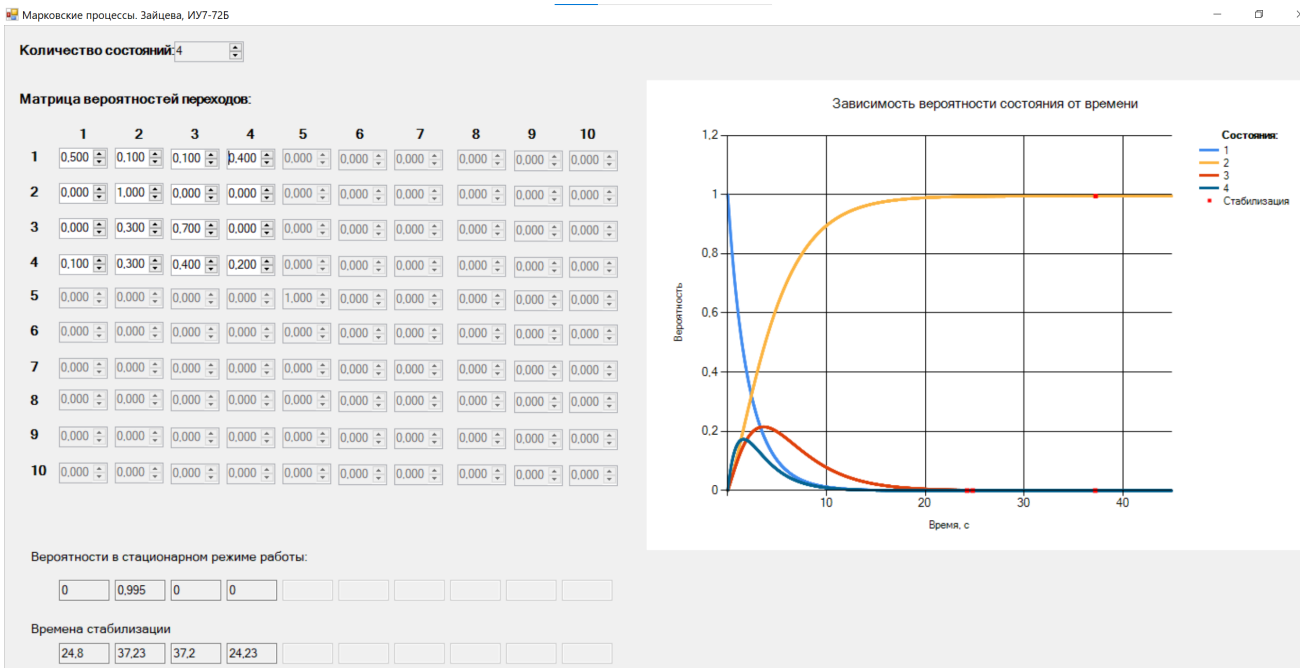

Рисунок 2 – Пример работы программы для 4 состояний

На рисунке 3 из первого состояния есть вероятность перейти в другие, но вероятность попасть из любого состояния в первое равна 0, поэтому и в стабилизировавшемся режиме вероятность первого состояния равна 0.

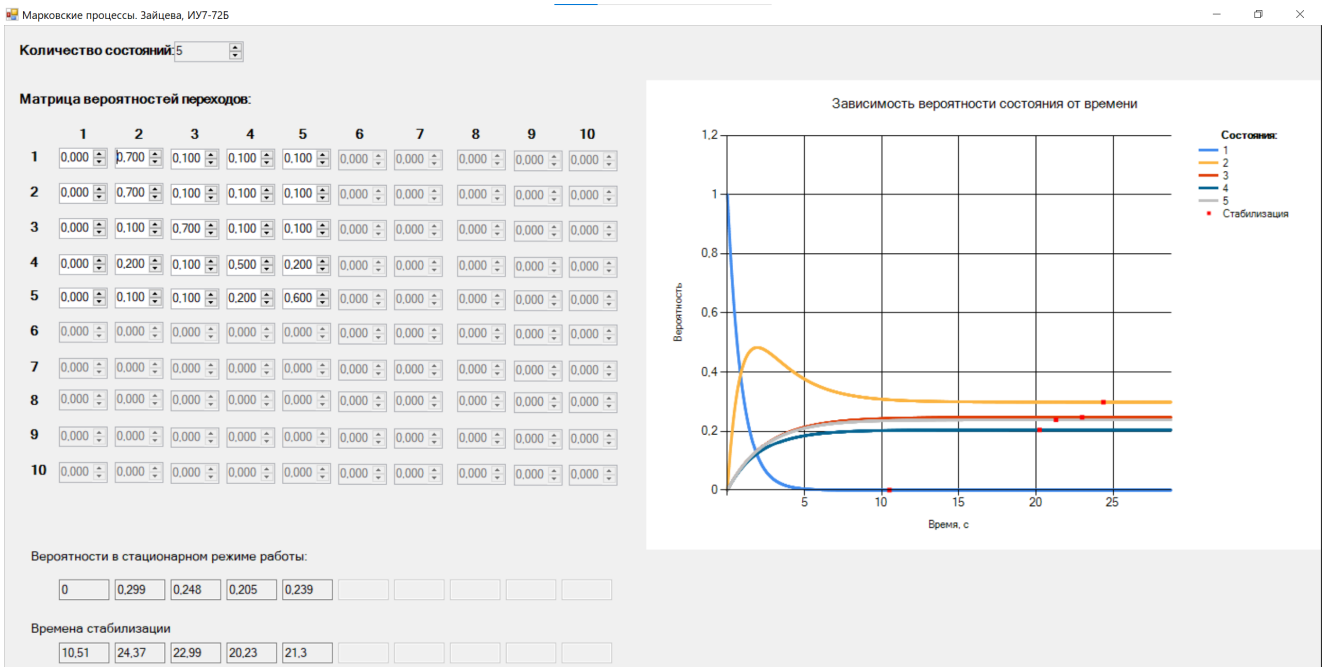

Рисунок 3 – Пример работы программы для 5 состояний

На рисунке 4 приведен пример работы программы для 10 состояний.

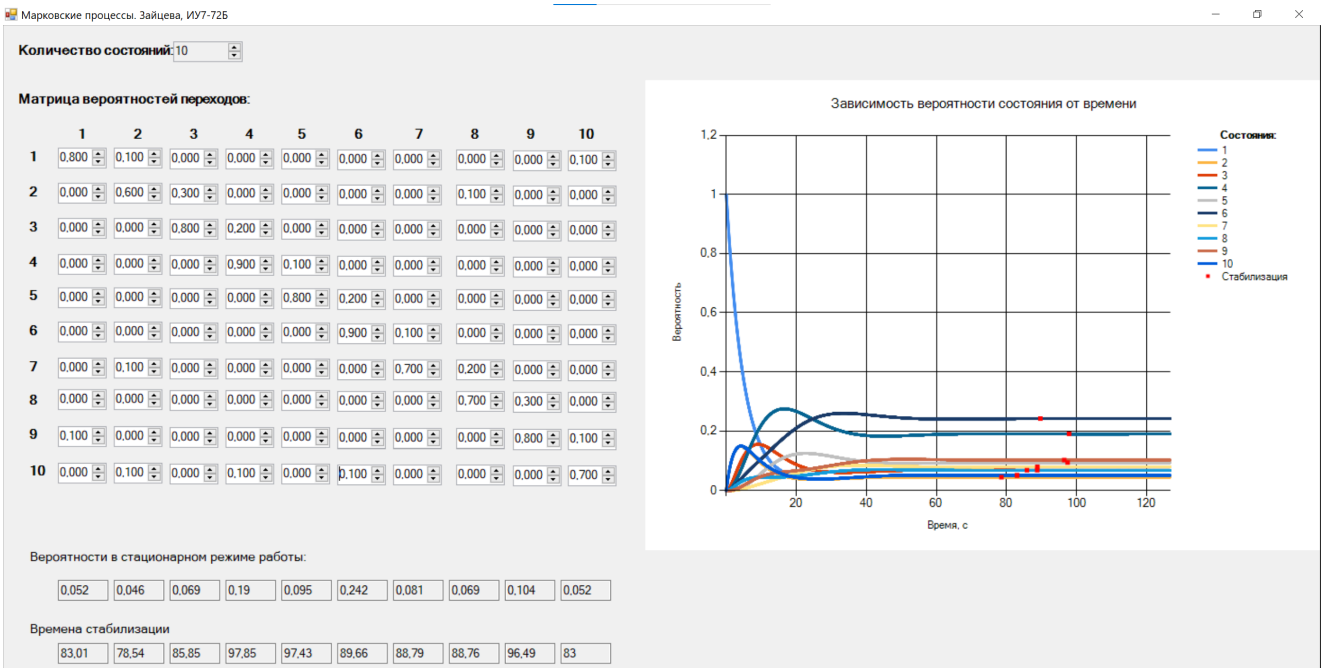

Рисунок 4 – Пример работы программы для 10 состояний

4 Код программы

Класс EmulationModel, используемый для расчетов и построения графиков, приведен в листинге 1 (используемый язык – C#).

Листинг 1 – Класс EmulationModel, используемый для расчетов и построения графиков

```
0     class EmulationModel
1 {
2     public int NStates;
3     public double[,] mtr;
4     public double[] pArr;
5     public double[] tStableArr;
6     public Chart currentChart;
7     readonly double step = 0.01;
8     readonly double stabEpsilon = 1e-5;
9     readonly double zeroEpsilon = 1e-8;
10
```

```
    public EmulationModel(int nStates, ref Chart chart)
    {
        NStates = nStates;
        pArr = new double[NStates];
        tStableArr = new double[NStates];
        mtr = new double[NStates, NStates];
        currentChart = chart;
        _initParray();
    }

    public void Emulate()
    {
        _initSeries();
        double[] deltaProbArray = new double[NStates];
        deltaProbArray[0] = 2 * stabEpsilon;

        for (double currentT = step; !_checkModelStabelized(deltaProbArray
); currentT += step)
        {
            _drawArrayOnCurrentT(currentT, pArr);

            deltaProbArray = new double[NStates];
            double[] PderivativeArr = new double[NStates];

            for (int i = 0; i < NStates; i++)
            {
                for (int j = 0; j < NStates; j++)
                {
                    double probDensityToAdd = mtr[j, i] * pArr[j] - mtr[i,
 j] * pArr[i];
                    PderivativeArr[i] += probDensityToAdd;
                    deltaProbArray[i] += probDensityToAdd * step;
                }
                pArr[i] += deltaProbArray[i];
            }

            _checkSomeStatesStabelized(currentT, PderivativeArr);
        }
        _drawStabelizedParr();
    }
```

```
49
50      private void _initParray()
51      {
52          pArr[0] = 1;
53          for (int i = 1; i < NStates; i++)
54          pArr[i] = 0;
55      }
56
57      private void _initSeries()
58      {
59          currentChart.Series.Clear();
60          for (int i = 0; i < NStates; i++)
61          {
62              currentChart.Series.Add((i + 1).ToString());
63              currentChart.Series[i].ChartType = SeriesChartType.Line;
64              currentChart.Series[i].BorderWidth = 3;
65          }
66
67          currentChart.Series.Add("Стабилизация");
68          currentChart.Series[NStates].ChartType = SeriesChartType.Point;
69          currentChart.Series[NStates].Color = Color.Red;
70      }
71
72      private bool _checkModelStabelized(double[] arr)
73      {
74          for (int i = 0; i < arr.Length; i++)
75          if (arr[i] > zeroEpsilon)
76          return false;
77          return true;
78      }
79
80      private void _checkSomeStatesStabelized(double currentT, double[]
    klmArr)
81      {
82          for (int i = 0; i < NStates; i++)
83          {
84              if (Math.Abs(klmArr[i]) < stabEpsilon && tStableArr[i] == 0)
85              tStableArr[i] = currentT;
86
87              else if (Math.Abs(klmArr[i]) > stabEpsilon && tStableArr[i] !=
```

```
    0)
88              tStableArr[i] = 0;
89          }
90      }
91
92      private void _drawArrayOnCurrentT(double currentT, double[] arr)
93      {
94          for (int i = 0; i < NStates; i++)
95          {
96              currentChart.Series[i].Points.AddXY(currentT, arr[i]);
97          }
98      }
99
100     private void _drawStabelizedParr()
101     {
102         for (int i = 0; i < NStates; i++)
103         {
104             currentChart.Series[NStates].Points.AddXY(tStableArr[i], pArr[
    i]);
105         }
106     }
107 }
```