|  |  |
| --- | --- |
| 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»

**ОТЧЕТ**

**по лабораторной работе № 2**

**по курсу: «Моделирование»**

|  |  |
| --- | --- |
| **Тема Марковские процессы**  **Студент Якуба Д. В.**  **Группа ИУ7-73Б**  **Оценка (баллы)**  **Преподаватель Рудаков И.В.** |  |

Москва, 2021

1. Задание

Написать программу, которая позволяет определить время пребывания случайной системы в каждом из состояний (при ). Количество состояний не более десяти. Для каждого состояния также рассчитать предельную вероятность.

2. Теория

Случайный процесс, протекающий в некоторой системе, называют марковским, если он обладает следующим свойством: для каждого момента времени вероятность любого состояния системы в будущем (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:

где – вероятность нахождения системы в состоянии в момент времени , – количество состояний в системе, – интенсивность перехода системы из состояния в состояние .

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:

Для определения точки стабилизации системы можно определять вероятности нахождения в определённых состояниях с некоторым малым шагом . 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: и , где может, например, принять значение .

3. Выполнение

На рисунке 2.1 предоставлен интерфейс разработанного приложения.

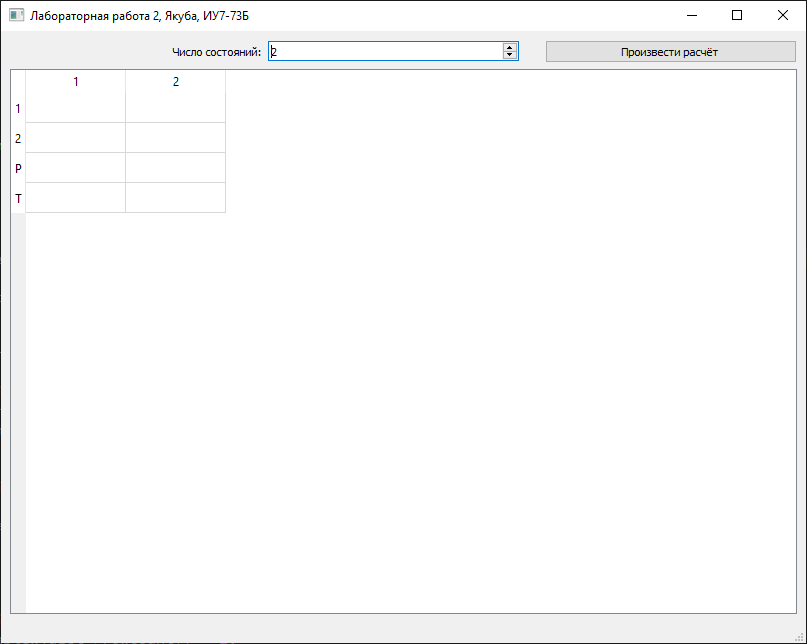


Рис. 2.1, Интерфейс приложения

3.1 Примеры работы

На рисунках 2.3-2.5 предоставлены примеры работы приложения.

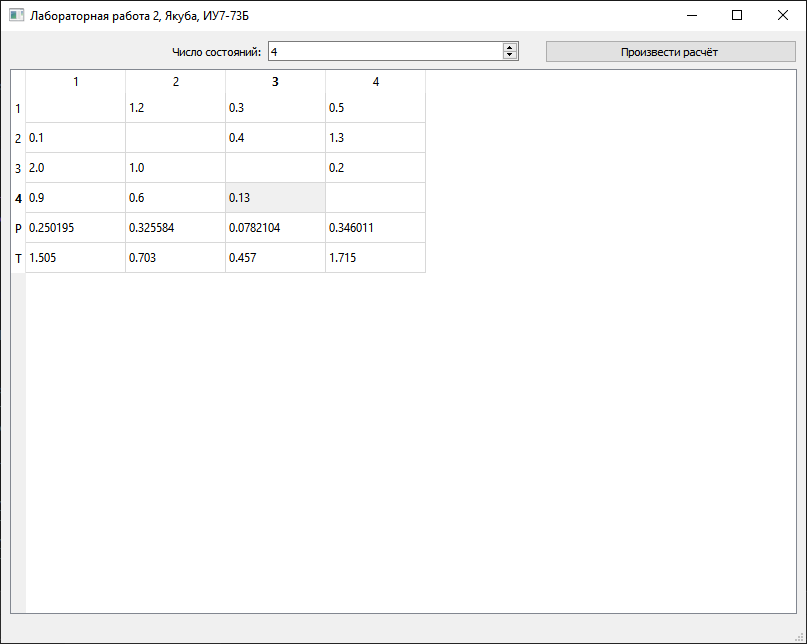


Рис. 2.2, Пример работы приложения для системы, включающей 4 состояния

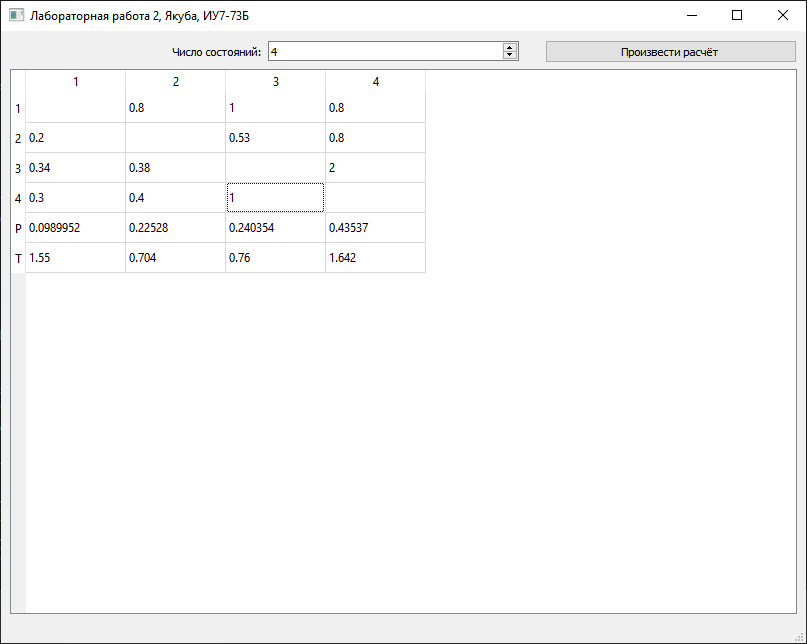


Рис. 2.3, Пример работы приложения для системы, включающей 4 состояния

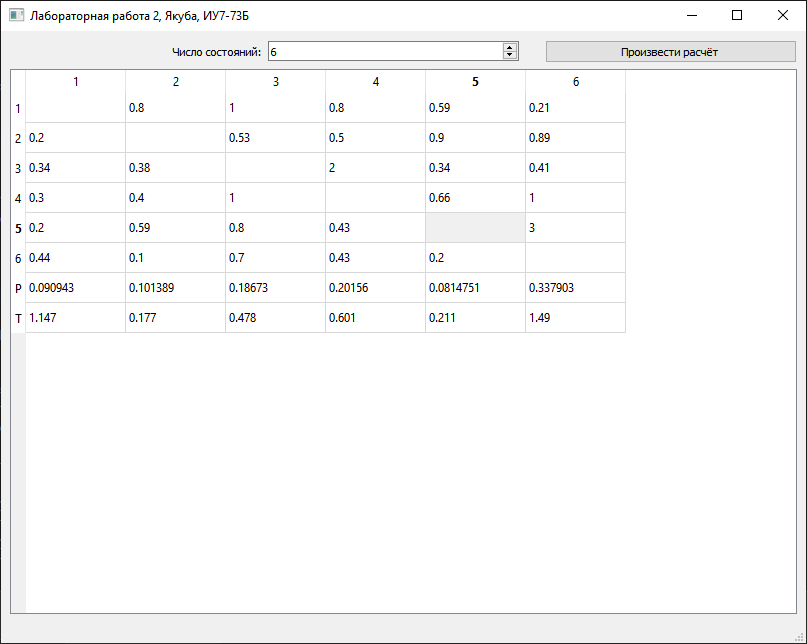


Рис. 2.4, Пример работы приложения для системы, включающей 6 состояний

4. Листинг

В данном разделе предоставлены используемые методы для решения поставленной задачи (используемый ЯП – C++).

|  |
| --- |
| QVector<QVector<double>> **buildSystemOfKolmogorovEquations**(  const QVector<QVector<double>> &intensityMatrix)  {  int numberOfStates = intensityMatrix.size();  QVector<QVector<double>> result(numberOfStates, QVector<double>(numberOfStates + 1));  for (int curState = 0; curState < numberOfStates - 1; curState++)  {  for (int col = 0; col < numberOfStates; col++)  { result[curState][curState] -= intensityMatrix[curState][col]; }  for (int row = 0; row < numberOfStates; row++)  { result[curState][row] += intensityMatrix[row][curState]; }  }  for (int state = 0; state < numberOfStates; state++)  { result[numberOfStates - 1][state] = 1; }  result[numberOfStates - 1][numberOfStates] = 1;  return result;  }  QVector<double> **getFundamentalDecisionSystem**(const QVector<QVector<double>> &intensityMatrix)  {  QVector<QVector<double>> systemOfKolmogorovEquations =  buildSystemOfKolmogorovEquations(intensityMatrix);  return gauss(systemOfKolmogorovEquations);  }  QVector<double> **probabilityDerivatives**(const QVector<QVector<double>> &intensityMatrix,  const QVector<double> &probabilities, double timeDelta)  {  int numberOfStates = intensityMatrix.size();  QVector<double> probabilityDerivatives(numberOfStates);  for (int i = 0; i < numberOfStates; i++)  {  double sumForProbability = 0;  for (int j = 0; j < numberOfStates; j++)  {  sumForProbability +=  probabilities[j] \*  ((i != j) ? intensityMatrix[j][i]  : (intensityMatrix[i][i] -  std::accumulate(intensityMatrix[i].begin(),  intensityMatrix[i].end(), 0.0)));  }  probabilityDerivatives[i] = sumForProbability \* timeDelta;  }  return probabilityDerivatives;  }  QVector<double> **determineTime**(const QVector<QVector<double>> &intensityMatrix,  const QVector<double> &systemSolvation, const QVector<double> &probabilityList,  double timeDelta, double eps)  {  int numberOfStates = intensityMatrix.size();  QVector<double> listOfTimes(numberOfStates);  QVector<double> probabilities = probabilityList;  bool endOfSearchCondition = false;  for (double curTime = timeDelta; !endOfSearchCondition && curTime < 1e3;  curTime += timeDelta)  {  endOfSearchCondition = true;  QVector<double> probabilityDerivative =  probabilityDerivatives(intensityMatrix, probabilities, timeDelta);  for (int i = 0; i < numberOfStates; i++)  {  endOfSearchCondition =  std::abs(systemSolvation[i] - probabilities[i]) <= eps &&  probabilityDerivative[i] <= eps;  if (endOfSearchCondition && listOfTimes[i] == 0.0)  {  listOfTimes[i] = curTime;  }  probabilities[i] += probabilityDerivative[i];  }  }  return listOfTimes;  } |