Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**Департамент анализа данных и машинного обучения**

**Отчет по контрольной работе по предмету «Математическое и имитационное моделирование»**

**Вариант №132**

Выполнил:

студент группы ЗБ-ПИ19-2

Малкеров Геннадий Александрович

Москва 2022

Задание №1.

На ввод подается файл примерно такого содержимого:

14

0,38 0,32 0 0 0,01 0 0,29 0 0 0 0 0 0 0

0 0,42 0,15 0 0 0,07 0,36 0 0 0 0 0 0 0

0 0 0,11 0 0 0,42 0 0,47 0 0 0 0 0 0

0 0 0,59 0,12 0 0,29 0 0 0 0 0 0 0 0

0,38 0 0 0 0,04 0,28 0 0 0,11 0,01 0,18 0 0 0

0 0 0 0 0,29 0,19 0,32 0 0 0,2 0 0 0 0

0 0,26 0 0 0 0 0,06 0 0,08 0,38 0,22 0 0 0

0 0 0 0,62 0 0 0 0,13 0 0,25 0 0 0 0

0 0 0 0 0 0 0 0 0,28 0 0 0,39 0 0,33

0 0 0 0 0 0,09 0,42 0 0 0,27 0,17 0 0,05 0

0 0 0 0 0,09 0,38 0,22 0 0 0 0,05 0,22 0,04 0

0 0 0 0 0 0 0 0 0,09 0 0,22 0,23 0,46 0

0 0 0 0 0 0 0 0 0,71 0 0,2 0 0,09 0

0 0 0 0 0 0 0 0 0,24 0 0 0 0,64 0,12

Решение:

1. Вероятность того, что за 5 шагов система перейдет из состояния 9 в состояние 5 — 0,024897572400000005;
2. Вероятности состояний системы спустя 5 шагов, если в начальный момент вероятность состояний были следующими A=(0,02;0,12;0,03;0,13;0,1;0,07;0,15;0,03;0,04;0,04;0,01;0,08;0,16;0,02)— 0,025504023722000002 0,080564581287999995 0,027970413387999996 0,012339291488000000 0,040528604249000004 0,104385463325000025 0,147872540738000019 0,017980282520000002 0,124767088697000025 0,111505179946000010 0,096789776687999998 0,088082042477000008 0,076020986655000011 0,045689724819000008;
3. Вероятность первого перехода за 6 шагов из состояния 6 в состояние 11— 0,06292087238700002;
4. Вероятность перехода из состояния 4 в состояние 7 не позднее чем за 8 шагов — 0,7013449861290781;
5. Среднее количество шагов для перехода из состояния 1 в состояние 10 — 13,53357365695391;
6. Вероятность первого возвращения в состояние 3 за 7 шагов — 0,011709756105259997;
7. Вероятность возвращения в состояние 13 не позднее чем за 7 шагов — 0,71974186390972;
8. Среднее время возвращения в состояние 11 — 9,904691507245127;
9. Установившиеся вероятности

[0,021962069337130158 0,06976697128885473 0,01572659371883861 0,005985801214301953 0,03583284997110684 0,0865286628787944 0,12860377369097745 0,00849597591707374 0,15785470299529206 0,09405135487302718 0,10096225604489874 0,10879874090654756 0,10623473353992223 0,059195513623234455 ]

Сумма вероятностей [1] единица.

Задание №1 исходный код

PS.Для более подробного просмотра исходного кода советую воспользоваться архивом прикреплённом к решению или ссылкой <https://disk.yandex.ru/d/1D4QP2VTVvCsxA>

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using MathNet.Numerics.LinearAlgebra;

namespace MatimProgram

{

public class Task1

{

private static Dictionary<Index, double> \_someMatrix = new();

public static void Make()

{

Console.WriteLine("Считываем исходную матрицу из файла!");

var matrix = ReadSourceMatrix();

Console.WriteLine("Матрица считана!");

matrix.PrintToConsole();

Console.WriteLine();

Console.WriteLine("Задание 1.");

Console.WriteLine("Возводим матрицу в степень 5");

var matrixOfFirst = matrix.Power(5);

Console.WriteLine($"Вероятность перехода [{matrixOfFirst[9 - 1, 5 - 1]}]");

Console.WriteLine();

Console.WriteLine("Задание 2.");

var arraySupport = GetSupportMatrix();

var res = arraySupport \* matrixOfFirst;

res.PrintToConsole(18);

Console.WriteLine();

Console.WriteLine("Задание 3.");

Console.WriteLine($"Вероятность первого перехода за 6 шагов из состояния 6 в состояние 11 равно [{Task3(matrix, 6, 11, 6)}]");

Console.WriteLine();

Console.WriteLine("Задание 4.");

Console.WriteLine($"Вероятность перехода из состояния 4 в состояние 7 не позднее чем за 8 равно [{Task4(matrix, 4, 7, 8)}]");

Console.WriteLine();

Console.WriteLine("Задание 5.");

Console.WriteLine($"Среднее количество шагов для перехода из состояния 1 в состояние 10 [{Task5(matrix, 1, 10, 10000)}]");

Console.WriteLine();

Console.WriteLine("Задание 6.");

\_someMatrix = new Dictionary<Index, double>();

Console.WriteLine($"Вероятность первого возвращения в состояние 3 за 7 шагов [{Task6(matrix, 3, 7)}]");

Console.WriteLine();

Console.WriteLine("Задание 7.");

\_someMatrix = new Dictionary<Index, double>();

Console.WriteLine($"Вероятность возвращения в состояние 13 не позднее чем за 7 шагов [{Task7(matrix, 13, 7)}]");

Console.WriteLine();

Console.WriteLine("Задание 8.");

\_someMatrix = new Dictionary<Index, double>();

Console.WriteLine($"Среднее время возвращение в состояние 11 [{Task8(matrix, 11)}]");

Console.WriteLine();

Console.WriteLine("Задание 9.");

var eMatr = Matrix<double>.Build.DenseDiagonal(matrix.RowCount, 1);

var matrM = matrix.Transpose() - eMatr;

// ReSharper disable once InconsistentNaming

var matrixM\_ = matrM.CopyToNew();

for (var i = 0; i < matrixM\_.RowCount; i++)

{

for (var j = 0; j < matrixM\_.ColumnCount; j++)

{

if (i == matrix.RowCount - 1)

{

matrixM\_[i, j] = 1;

}

}

}

var bMatrix = Support.GetEmptyMatrix(matrix.RowCount, 1);

bMatrix[matrix.RowCount - 1, 0] = 1;

var inv = matrixM\_.Inverse();

var x = inv \* bMatrix;

var xNormal = new double[matrix.RowCount];

for (var i = 0; i < x.RowCount; i++)

{

for (var j = 0; j < x.ColumnCount; j++)

{

if (j == x.ColumnCount - 1)

xNormal[i] = x[i, j];

}

}

xNormal.PrintArray();

var result = xNormal.Sum();

Console.WriteLine($"Сумма вероятностей [{result}] единица");

}

private static double Task8(Matrix<double> sourse, int desteny)

{

var buffer = new List<double>();

for (var i = 1; i < 1000; i++)

{

buffer.Add(i \* Task6(sourse, desteny, i));

}

return buffer.Sum();

}

private static double Task7(Matrix<double> sourse, int desteny, int n)

{

var buffer = new List<double>();

for (var i = 1; i < n + 1; i++)

{

buffer.Add(Task6(sourse, desteny, i));

}

return buffer.Sum();

}

private static double Task6(Matrix<double> sourse, int start, int n)

{

var resultM = Support.GetEmptyMatrix(sourse.RowCount, sourse.ColumnCount);

for (var i = 1; i < n; i++)

{

double temp;

var ind = new Index()

{

Start = start,

I = i

};

if (!\_someMatrix.ContainsKey(ind))

{

temp = Task6(sourse, start, i);

\_someMatrix.Add(ind, temp);

}

else

{

temp = \_someMatrix[ind];

}

resultM += temp \* sourse.Power(n - i);

}

resultM = sourse.Power(n) - resultM;

return resultM[start - 1, start - 1];

}

private static double Task5(Matrix<double> sourse, int start, int end, int iteration)

{

var resultMatr = sourse.CopyToNew();

var result = resultMatr[start - 1, end - 1];

for (var i = 2; i < iteration; i++)

{

resultMatr = Task3(sourse, resultMatr);

result += (i \* resultMatr[start - 1, end - 1]);

}

return result;

}

private static double Task4(Matrix<double> sourse, int start, int end, int n)

{

var previousM = sourse.CopyToNew();

var resultM = sourse.CopyToNew();

var tempMatr = sourse.CopyToNew();

for (var i = 1; i < n; i++)

{

previousM = Task4(tempMatr, previousM);

resultM += previousM;

}

return resultM[start - 1, end - 1];

}

private static Matrix<double> Task4(Matrix<double> curentM, Matrix<double> previosM)

{

var resMatrix = Matrix<double>.Build.Dense(curentM.RowCount, curentM.ColumnCount);

for (var i = 0; i < curentM.RowCount; i++)

{

for (var j = 0; j < curentM.ColumnCount; j++)

{

var bufList = new List<double>();

for (var m = 0; m < curentM.RowCount; m++)

{

if (m != j)

{

bufList.Add(curentM[i, m] \* previosM[m, j]);

}

}

resMatrix[i, j] = bufList.Sum();

}

}

return resMatrix;

}

private static double Task3(Matrix<double> sourse, int start, int end, int n)

{

var tempMatrix = sourse.CopyToNew();

for (var i = 2; i < n + 1; i++)

{

tempMatrix = Task3(sourse, tempMatrix);

}

return tempMatrix[start - 1, end - 1];

}

private static Matrix<double> Task3(Matrix<double> sourse, Matrix<double> support)

{

var tempMatrix = Matrix<double>.Build.Dense(sourse.RowCount, sourse.ColumnCount);

for (var i = 0; i < sourse.RowCount; i++)

{

for (var j = 0; j < sourse.ColumnCount; j++)

{

var buffer = new List<double>();

for (var m = 0; m < sourse.RowCount; m++)

{

var buf = 0.0;

if (m != j)

{

buf = sourse[i, m] \* support[m, j];

}

buffer.Add(buf);

}

tempMatrix[i, j] = buffer.Sum();

}

}

return tempMatrix;

}

private static Matrix<double> ReadSourceMatrix()

{

var filePath = Path.Combine(Directory.GetCurrentDirectory(), "1taskData.txt");

if (!File.Exists(filePath))

throw new Exception($"Не удалось найти файл по пути [{filePath}]");

var allLines = File.ReadAllLines(filePath);

var length = int.Parse(allLines[0]);

var resMatrix = Matrix<double>.Build.Dense(length, length);

var index = 0;

for (var i = 1; i < allLines.Length; i++)

{

var listOf = allLines[i].Split("\t").Select(double.Parse).ToArray();

if (listOf.Length != length)

throw new Exception(

$"Не удалось распарсить данные из строки [{i + 1}] " +

$"количество распаршенных данных больше [{length}] и равно [{listOf.Length}]");

for (var j = 0; j < length; j++)

{

resMatrix[index, j] = listOf[j];

}

index++;

}

return resMatrix;

}

private static Matrix<double> GetSupportMatrix()

{

var res = Matrix<double>.Build.Dense(1, 14);

res[0, 0] = double.Parse("0,02");

res[0, 1] = double.Parse("0,12");

res[0, 2] = double.Parse("0,03");

res[0, 3] = double.Parse("0,13");

res[0, 4] = double.Parse("0,1");

res[0, 5] = double.Parse("0,07");

res[0, 6] = double.Parse("0,15");

res[0, 7] = double.Parse("0,03");

res[0, 8] = double.Parse("0,04");

res[0, 9] = double.Parse("0,04");

res[0, 10] = double.Parse("0,01");

res[0, 11] = double.Parse("0,08");

res[0, 12] = double.Parse("0,16");

res[0, 13] = double.Parse("0,02");

return res;

}

}

}

Задание 2.

Задана система массового обслуживания со следующими характеристиками:

• интенсивность поступления λ=14

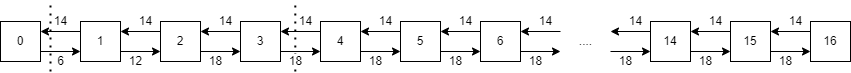
• каналов обслуживания m=3

• интенсивность обслуживания μ=6

• максимальный размер очереди n=13

a) Составьте граф марковского процесса, запишите систему уравнений Колмогорова и найдите установившиеся вероятности состояний.

Граф Марковского процесса:



Система уравнений Колмогорова:

Изображение выглядит как текст

Автоматически созданное описание

Вероятности состояний:

[0,06535578017249621 0,15249682040249116 0,17791295713623967 0,1383767444392975 0,10762635678612027 0,08370938861142688 0,06510730225333201 0,050639012863702676 0,03938589889399098 0,030633476917548537 0,02382603760253775 0,01853136257975158 0,014413282006473451 0,011210330449479352 0,008719145905150612 0,006781557926228257 0,005274545053733091 ];

b) Найдите вероятность отказа в обслуживании — 0,005274545053733091;

c) Найдите относительную и абсолютную интенсивность обслуживания — Относительная пропускная способность [0,994725454946267]

Абсолютная пропускная способность [13,926156369247737];

d) Найдите среднюю длину в очереди — 1,8563678403777841;

e) Найдите среднее время в очереди — 0,13259770288412742;

f) Найдите среднее число занятых каналов — 2,321026061541289;

g) Найдите вероятность того, что поступающая заявка не будет ждать в очереди — 0.3957655577112281;

h) Найти среднее время однократного простоя системы массового обслуживания — 0,07142857142857142.

Задание №2 исходный код

PS.Для более подробного просмотра исходного кода советую воспользоваться архивом прикреплённом к решению или ссылкой <https://disk.yandex.ru/d/1D4QP2VTVvCsxA>

using System;

using System.Collections.Generic;

using System.Linq;

using MathNet.Numerics.LinearAlgebra;

namespace MatimProgram

{

public class Task2

{

private readonly double \_l = 14.0;

private readonly double \_m = 3.0;

private readonly double \_u = 6.0;

private readonly double \_n = 13.0;

private readonly int \_mInt = 3;

private readonly int \_uInt = 6;

private readonly int \_nInt = 13;

public void Make()

{

var matrixNew = Matrix<double>.Build.Dense(\_mInt + \_nInt + 1, \_mInt + \_nInt + 1);

for (var i = 0; i < \_mInt + \_nInt; i++)

{

matrixNew[i, i + 1] = \_l;

if (i < \_mInt)

matrixNew[i + 1, i] = (\_uInt \* (i + 1));

else

matrixNew[i + 1, i] = \_u \* \_m;

}

Console.WriteLine("Задание А.");

Console.WriteLine("Матрица интенсивности переходов");

matrixNew.PrintToConsole();

var matrixSumm = new double[\_mInt + \_nInt + 1];

for (var i = 0; i < matrixNew.RowCount; i++)

{

var sum = 0.0;

for (var j = 0; j < matrixNew.ColumnCount; j++)

{

sum += matrixNew[i, j];

}

matrixSumm[i] = sum;

}

var matrixD = Matrix<double>.Build.Dense(\_mInt + \_nInt + 1, \_mInt + \_nInt + 1);

for (var i = 0; i < matrixD.RowCount; i++)

{

for (var j = 0; j < matrixD.ColumnCount; j++)

{

if (i == j)

{

matrixD[i, j] = matrixSumm[i];

}

}

}

var matrixM = matrixNew.Transpose() - matrixD;

// ReSharper disable once InconsistentNaming

var matrixM\_ = matrixM.CopyToNew();

for (var i = 0; i < matrixM\_.RowCount; i++)

{

for (var j = 0; j < matrixM\_.ColumnCount; j++)

{

if (i == \_mInt + \_nInt)

{

matrixM\_[i, j] = 1;

}

}

}

var bMatrix = Matrix<double>.Build.Dense(\_mInt + \_nInt + 1, 1);

bMatrix[bMatrix.RowCount - 1, 0] = 1;

var inv = matrixM\_.Inverse();

var x = inv \* bMatrix;

var xNormal = new double[\_mInt + \_nInt + 1];

for (var i = 0; i < x.RowCount; i++)

{

for (var j = 0; j < x.ColumnCount; j++)

{

if (j == x.ColumnCount - 1)

xNormal[i] = x[i, j];

}

}

xNormal.PrintArray();

var res = xNormal.Sum();

Console.WriteLine($"Сумма вероятностей [{res}] почти еденица, но списать можно на погрешность ибо числа слишком маленькие");

Console.WriteLine();

Console.WriteLine("Задание B.");

Console.WriteLine($"Вероятность отказа от обслуживания [{xNormal[\_nInt + \_mInt]}]");

Console.WriteLine();

Console.WriteLine("Задание C.");

Console.WriteLine($"Отсносительная пропускная способность [{1 - xNormal[\_nInt + \_mInt]}]");

Console.WriteLine($"Абсолютная пропускная способность [{(1 - xNormal[\_nInt + \_mInt]) \* \_l}]");

Console.WriteLine();

Console.WriteLine("Задание D.");

var resOfLength = new List<double>();

for (var i = 1; i < \_n + 1; i++)

{

resOfLength.Add(i \* xNormal[\_mInt + i]);

}

Console.WriteLine($"Средняя длина очереди [{resOfLength.Sum()}]");

Console.WriteLine();

Console.WriteLine("Задание E.");

resOfLength = new List<double>();

for (var i = 0; i < \_nInt; i++)

{

resOfLength.Add((i + 1.0) / (\_mInt \* \_uInt) \* xNormal[\_mInt + i]);

}

Console.WriteLine($"Среднее время в очереди [{resOfLength.Sum()}]");

Console.WriteLine();

Console.WriteLine("Задание F.");

var resOfLeft = new List<double>();

var resOfRight = new List<double>();

for (var i = 0; i < \_m + 1; i++)

{

resOfLeft.Add(i \* xNormal[i]);

}

for (var i = \_mInt + 1; i < \_mInt + \_nInt + 1; i++)

{

resOfRight.Add(\_m \* xNormal[i]);

}

Console.WriteLine($"Сумма левой части [{resOfLeft.Sum()}]");

Console.WriteLine($"Сумма правой части [{resOfRight.Sum()}]");

Console.WriteLine($"Среднее число занятых каналов [{resOfLeft.Sum() + resOfRight.Sum()}]");

Console.WriteLine();

Console.WriteLine("Задание G.");

var resOf = new List<double>();

for (var i = 0; i < \_m; i++)

{

resOf.Add(xNormal[i]);

}

Console.WriteLine($"Вероятность того, что поступающая заявка не будет ждать в очереди [{resOf.Sum()}]");

Console.WriteLine();

Console.WriteLine("Задание H.");

Console.WriteLine($"Среднее время простоя системы массового обслуживания [{1.0 / \_l}]");

}

}

}