Національний Технічний Університет України

«Київський Політехнічний Інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №5

|  |  |
| --- | --- |
| Прийняв  Доц. Марковський О.П.  «\_\_»\_\_\_\_\_\_\_\_ 2015 р. | Виконав  Студент 2-ого курсу ФІОТ  групи ІО-41  Смішний Д.М. |

**Варіант завдання**:

ПМ – 3 2 стани Неперервний

**Код програми**

**public** **class** WorkClass {

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) {

**double**[][] semiGraph = { { 0, 1 },

{ 2, 1 },

};

MakeMarkovGraph graph = MakeMarkovGraph.*fromSemiMarkovGraph*(semiGraph);

**double**[] res = graph.simulate(1000);

System.***out***.println("Experimental part :");

**double** sum = 0;

**for**(**int** i = 0; i < res.length; i++) {

sum += res[i];

System.***out***.println(res[i]);

}

System.***out***.println(sum);

System.***out***.println("Theoretical part :");

res = graph.modeling();

sum = 0;

**for**(**int** i = 0; i < res.length; i++) {

sum += res[i];

System.***out***.println(res[i]);

}

System.***out***.println(sum);

}

}

**public** **class** MakeMarkovGraph {

**public** **static** MakeMarkovGraph fromSemiMarkovGraph(**double**[][] semiMarkovGraph) {

**int** size = semiMarkovGraph.length;

**int** index = -1;

**int**[] trueVerticies = **new** **int**[size];

**for** (**int** i = 0; i < semiMarkovGraph.length; i++) {

trueVerticies[i] = ++index;

**for** (**int** j = 0; j < semiMarkovGraph.length; j++) {

**if**(semiMarkovGraph[i][j] != 0) {

size++;

index++;

}

}

}

**int** col = 0;

**int** row = 0;

**double**[][] adjacencyMatrix = **new** **double**[size][size];

**for** (**int** i = 0; i < semiMarkovGraph.length; i++) {

col = trueVerticies[i];

row = trueVerticies[i];

**for** (**int** j = 0; j < semiMarkovGraph.length; j++) {

**if** (semiMarkovGraph[i][j] != 0) {

adjacencyMatrix[row][col + 1] = semiMarkovGraph[i][j] \* 2;

adjacencyMatrix[col + 1][trueVerticies[j]] = semiMarkovGraph[i][j] \* 2;

col++;

}

}

}

**return** **new** MakeMarkovGraph(adjacencyMatrix, trueVerticies);

}

**private** **int**[] trueVertices;

**private** **double**[][] adjacencyMatrix;

**private** **double** timeToFirstVertices;

**private** **double** T;

**private** MakeMarkovGraph(**double**[][] adjacencyMatrix, **int**[] trueVertices) {

**this**.adjacencyMatrix = adjacencyMatrix;

**this**.trueVertices = trueVertices;

timeToFirstVertices = 0;

}

**public** **double**[] simulate(**int** capasity) {

**double**[] time = **new** **double**[adjacencyMatrix.length];

**double** totalTime = 0;

**int** nextVertice = 0;

**int** currentVertice = 0;

**double**[] transitionTime;

**for** (**int** i = 0; i < capasity; i++) {

transitionTime = calcTransitionTime(currentVertice);

nextVertice = getMinIndex(transitionTime);

**if**(nextVertice != 0) {

timeToFirstVertices += transitionTime[nextVertice];

}

time[currentVertice] += transitionTime[nextVertice];

totalTime += transitionTime[nextVertice];

currentVertice = nextVertice;

}

**double**[] timeInTrueVertices = **new** **double**[trueVertices.length];

**int** indexOfTrueVertice = 0;

**for** (**int** i = 0; i < time.length; i++) {

timeInTrueVertices[indexOfTrueVertice] += time[i];

**if** (indexOfTrueVertice < trueVertices.length - 1

&& i == trueVertices[indexOfTrueVertice + 1] - 1) {

indexOfTrueVertice++;

}

}

T = totalTime;

**for** (**int** i = 0; i < timeInTrueVertices.length; i++) {

timeInTrueVertices[i] /= totalTime;

}

**return** timeInTrueVertices;

}

**private** **double**[] calcTransitionTime(**int** verticeNum) {

Random random = **new** Random();

**double**[] res = **new** **double**[adjacencyMatrix[verticeNum].length];

**for** (**int** i = 0; i < res.length; i++) {

**if** (adjacencyMatrix[verticeNum][i] != 0) {

**double** r = random.nextDouble();

res[i] = - 1 / adjacencyMatrix[verticeNum][i] \* Math.*log*(random.nextDouble());

}

}

**return** res;

}

**private** **int** getMinIndex(**double**[] arr) {

**int** minIndex = 0;

**for** (**int** i = 1; i < arr.length; i++) {

**if**(arr[minIndex] == 0

|| (arr[i] !=0 && arr[minIndex] > arr[i])) {

minIndex = i;

}

}

**return** minIndex;

}

**public** **double**[][] buildEquations() {

**double**[][] equation = **new** **double**[adjacencyMatrix.length][adjacencyMatrix.length + 1];

**for** (**int** i = 0; i < equation[adjacencyMatrix.length - 1].length; i++) {

equation[adjacencyMatrix.length - 1][i] = 1;

}

**for** (**int** i = 0; i < adjacencyMatrix.length - 1; i++) {

**int** sum = 0;

**for** (**int** j = 0; j < adjacencyMatrix.length; j++) {

**if** (adjacencyMatrix[i][j] != 0) {

sum += adjacencyMatrix[i][j];

}

equation[i][i] = - sum;

**if** (adjacencyMatrix[j][i] != 0) {

equation[i][j] = adjacencyMatrix[j][i];

}

}

}

**return** equation;

}

**public** **double**[] modeling() {

**double**[] arr = MatrixWorking.*getSolutionOfEquations*(buildEquations());

**double**[] timeInTrueVertices = **new** **double**[trueVertices.length];

**int** indexOfTrueVertice = 0;

**for** (**int** i = 0; i < arr.length; i++) {

timeInTrueVertices[indexOfTrueVertice] += arr[i];

**if** (indexOfTrueVertice < trueVertices.length - 1

&& i == trueVertices[indexOfTrueVertice + 1] - 1) {

indexOfTrueVertice++;

}

}

**return** timeInTrueVertices;

}

**public** **double** getTimeToFirstVertices() {

**return** timeToFirstVertices / T;

}

@Override

**public** String toString() {

StringBuilder builder = **new** StringBuilder();

**for** (**int** i = 0; i < adjacencyMatrix.length; i++) {

**for** (**int** j = 0; j < adjacencyMatrix.length; j++) {

builder.append(String.*format*("%10.3f", adjacencyMatrix[i][j]));

}

builder.append("\n");

}

builder.append("\n");

builder.append("True verticies: ");

**for** (**int** i = 0; i < trueVertices.length; i++) {

builder.append(String.*format*("%3d", trueVertices[i]));

}

**return** builder.toString();

}

}

**public** **class** MatrixWorking {

**public** **static** **double**[] getSolutionOfEquations(**double**[][] a) {

**double**[][] x = **new** **double**[a.length][a[0].length];

**for** (**int** i = 0; i < a.length; i++)

x[i] = Arrays.*copyOf*(a[i], a[i].length);

**double** p;

**for** (**int** i = 0; i < x.length; i++)

**for** (**int** j = 0; j < x.length; j++)

**if** (i != j) {

p = x[j][i] / x[i][i];

**for** (**int** k = i; k <= x.length; ++k)

x[j][k] -= p \* x[i][k];

}

**double**[] b = **new** **double**[x.length];

**for** (**int** i = 0; i < x.length; i++)

b[i] = x[i][x.length] / x[i][i];

**return** b;

}

}

**Результати:**

Experimental part :

0.5738733077505026

0.42612669224949784

Theoretical part :

0.5714285714285714

0.4285714285714286