Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №5

З предмету: «Теорія ймовірності»

варіант – ПМ-3

Виконав: студент групи ІО-22

Щербина Микита

Київ 2013р.

Граф переходів



Результати

λ12=2

λ21=3

Теоретические значения

P1 =p1+p11+p12= 0.5999999999999996

P2 =p2+p21+p22= 0.39999999999999974

Практические значения

P1 = p1+p11+p12 = 0.5655922062013017

P2 = p2+p21+p22 = 0.4344077937986999

**import** java.util.Arrays;

**import** java.util.Scanner;

**import** org.apache.commons.math3.linear.\*;

**public** **class** Main {

**static** **int** *z*[][][] = {

{ { 1, 0 }, { 2, 1 }, { 3, 2 } },

{ { 0, 4 }, { 4, 5 }, { 5, 3 } } };

**static** **int** *countEquations* = 6;

/\*\*

\* решение системы уравнений

\*

\* @param matrix

\* - матрица коефициентов

\* @param resultRow

\* - столбец с результатами уравнений

\* @return - столбец с решением системы

\*/

public static double[] solution(double[][] matrix, double[] resultRow) {

RealMatrix realMatrix = new Array2DRowRealMatrix(matrix);

RealMatrix realResultRow = new Array2DRowRealMatrix(resultRow);

RealMatrix inverseMatrix = MatrixUtils.blockInverse(realMatrix, 1);

RealMatrix result = inverseMatrix.multiply(realResultRow);

result = result.transpose();

return result.getData()[0];

}

/\*\*

\* создает матрицу системы уравнений по матрице графа

\*

\* @param g

\* - матрица графа

\* @return

\*/

static public double[][] createMatrixEquations(double[][] g) {

double[][] result = new double[g.length][countEquations];

Arrays.fill(result[result.length - 1], 1.0);

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

for (int j = 0; j < g[i].length; j++)

if (g[i][j] < 0) {

result[i][i] += g[i][j];

} else

result[i][j] += g[i][j];

}

return result;

}

public static void main(String[] args) {

double[] teoryP = new double[2];

double[] modelP = new double[2];

double[][] a = new double[][] { {1, 10}, {8, 9} };

// матрица с описанием графа

double[][] g = new double[countEquations][countEquations];

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

for (int j = 0; j < z[i].length; j++) {

g[z[i][j][0]][z[i][j][1]] = 3 \* a[i/2][i%2];

g[z[i][j][1]][z[i][j][0]] = -3 \* a[i/2][i%2];

}

}

// матрица уравнений

double[][] matrixEquations = createMatrixEquations(g);

double[] resultRow = new double[countEquations];

resultRow[resultRow.length - 1] = 1;

double[] p = solution(matrixEquations, resultRow);

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

if (j < 5)

teoryP[0] += p[j];

else

teoryP[1] += p[j];

}

System.out.println("Теоретические значения");

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

System.out.println("p" + (i + 1) + " = " + teoryP[i]);

}

int count = 1000;

MarkovProcess mp = new MarkovProcess(g, 0);

for (int i = 0; i < count; i++) {

try {

mp.next();

} catch (StopException e) {

break;

}

}

System.out.println();

double[] mP = mp.pState();

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

if (i < 5)

modelP[0] += mP[i];

else

modelP[1] += mP[i];

}

System.out.println("Практические значения");

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

System.out.println("p" + (i + 1) + " = " + modelP[i]);

}

}

}  
**import** java.util.Random;

**public** **class** MarkovProcess {

**private** **double** eps = -0.001;

**private** Random random;

**private** **int** countState;

**private** **int** currentState;

**private** **double**[][] g;

**public** **double**[] timeState;

**public** **double** timeSystem;

**public** MarkovProcess(**double**[][] g, **int** startState) {

**this**.g = g;

random = **new** Random();

countState = g.length;

timeState = **new** **double**[countState];

currentState = startState;

timeSystem = 0;

}

/\*\*

\* переход в следуеще состояние

\*

\* **@throws** StopException

\*/

**public** **void** next() **throws** StopException {

**double**[] nextTimeState = **new** **double**[countState];

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

**if** (g[currentState][i] < eps)

nextTimeState[i] = Math.*log*(random.nextDouble())/g[currentState][i];

**else**

nextTimeState[i] = Double.*MAX\_VALUE*;

}

**double** minTime = Double.*MAX\_VALUE*;

**int** minIndex = -1;

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

**if** (nextTimeState[i] < minTime) {

minTime = nextTimeState[i];

minIndex = i;

}

}

**if** (minIndex == -1) {

finalState(currentState);

**throw** **new** StopException();

}

timeState[currentState] += minTime;

timeSystem += minTime;

currentState = minIndex;

}

**private** **void** finalState(**int** state){

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

timeState[i]=0;

}

timeState[state]=timeSystem;

}