*Національний технічний університет України*

*«Київський політехнічний інститут»*

***Факультет інформатики та обчислювальної техніки***

Лабораторна робота №5

*з курсу "****Теорія ймовірностей****"*

**Тема: “Марковські процеси”**

***Виконав:***

*Долинний О.В.*

***Група*** *ІО-31*

***Перевірив:***

*Марковський О.П.*

***Київ - 2014р.***

**Завдання: ПМ-3, 2 вершини, неперервний**

**Клас Main**

**package** lab\_5;

**import** java.text.DecimalFormat;

**import** java.text.DecimalFormatSymbols;

**import** java.util.Random;

/\*\*

\* Created by Admin on 11/20/2014.

\*/

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

**private** **static** **int** *numberOfStates* = 2;

**private** **static** **int** *start* = 0;

**private** **static** **int** *stages* = 3;

**private** **static** **int** *currentState* = *start*;

**private** **static** **double**[] *timeIn* = **new** **double**[*numberOfStates*];

**private** **static** **double** *timeAll* = 0;

**private** **static** **int** *numberOfJumps* = 100000;

**private** **static** **double**[] *staticProb* = **new** **double**[*numberOfStates*];

**private** **static** **double**[][] *matrixOfIntens* = {{1, 1}, {0.5, 0.5}};

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

**private** **static** **int**[] *entry* = **new** **int**[*numberOfStates*];

**private** **static** DecimalFormatSymbols *symbols* = **new** DecimalFormatSymbols();

**private** **static** DecimalFormat *format* = **new** DecimalFormat("#,##0.000", *symbols*);

**private** **static** **double**[] *teoryResult* = **new** **double**[*numberOfStates*];

**private** **static** **double** *specialFor1* = 0;

**private** **static** **double** *specialFor2* = 0;

**private** **static** **double** *timeForBack* = 0;

**private** **static** **int** *countForBack* = 0;

**private** **static** **boolean** *flag* = **true**;

/\*\*

\* here is the main function

\* **@param** args

\*/

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

*getTeoryResult*();

System.***out***.println("Theory results are: ");

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

System.***out***.println(*format*.format(*teoryResult*[i]));

}

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

*makeStep*();

}

*getStaticProb*();

//System.out.println("All time are: " + format.format(timeAll));

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

System.***out***.println("Static time in " + i + "'st state is: " + *format*.format(*staticProb*[i]));

//System.out.println("here entry was: " + entry[i]);

}

/\*for (int i = 0; i < numberOfStates; i++) {

System.out.println("Average time in" + i + ": "+ staticProb[i]\*timeAll/entry[i]);

}

System.out.println(timeForBack / countForBack);

\*/

}

**public** **static** **double** getRandTime(**double** intensity){

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

**return** -Math.*log*(prob) / intensity;

}

**public** **static** **double** getRandToNext(**double** intensity){

**double** time = 0;

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

time += *getRandTime*(intensity);

}

**return** time;

}

**public** **static** **void** makeStep(){

**double**[] choice = **new** **double**[*numberOfStates*];

**double**[] lineFrom = *matrixOfIntens*[*currentState*];

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

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

**if** (lineFrom[i] > 0){

choice[i] = *getRandToNext*(lineFrom[i]);

**if** (choice[i] < min)

min = choice[i];

}

**else**{

choice[i] = Double.***POSITIVE\_INFINITY***;

}

}

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

**if** (choice[i] == min){

**if** (*currentState* == 0 && *flag*){

*flag* = **false**;

*countForBack*++;

}

**else** {

**if** (!*flag*) {

*timeForBack* += min;

}

**if** (i == 0){

*flag* = **true**;

}

}

*entry*[i]++;

*timeIn*[*currentState*] += min;

*currentState* = i;

**break**;

}

}

*updateTime*();

}

**public** **static** **void** updateTime(){

*timeAll* = 0;

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

*timeAll* +=*timeIn*[i];

}

}

**public** **static** **void** getStaticProb(){

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

*staticProb*[i] = *timeIn*[i] / *timeAll*;

}

}

**public** **static** **void** getTeoryResult(){

//first we generate our matrix for computation

**double** [][] sysEq = **new** **double**[*numberOfStates*][*numberOfStates* + 1];

**for** (**int** j = 0; j < *numberOfStates* + 1; j++) {

sysEq[0][j] = 1;

}

**for** (**int** i = 1; i < *numberOfStates*; i++)

**for** (**int** j = 0; j < *numberOfStates*; j++) {

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

**for** (**int** k = 0; k < *numberOfStates*; k++)

sysEq[i][j] += *matrixOfIntens*[i][k];

sysEq[i][j] -= *matrixOfIntens*[i][j];

}

**else**

sysEq[i][j] = -*matrixOfIntens*[j][i];

}

EquationsSet equationsSet = **new** EquationsSet();

equationsSet.Coefficients = sysEq;

//equationsSet.print();

*teoryResult* = equationsSet.Solve();

}

}

**Клас EquationsSet**

**package** lab\_5;

**public** **class** EquationsSet {

**public** **double**[][] Coefficients; // Рядок - колонка

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

// Фактично є проксі-методом, що переадресовує запит до рекурсивної ф-ї

// розв'язання системи, а потім повертає результат. Так зроблено для

// зручності введення рекурсії

**this**.Simplify();

**this**.MakeNullsInUpperLines();

**double**[][] t = Coefficients;

**double**[] Result = **new** **double**[t.length];

// У результат переписую вільні члени після розв'язання

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

Result[i] = t[i][t[i].length - 1];

}

**return** Result;

}

/\*\*

\* Виводить на екран коефіцієнти системи

\*/

**public** **void** print() {

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

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

System.***out***.printf("%-7f ", Coefficients[i][j]);

}

System.***out***.println(" = "

+ Coefficients[i][Coefficients[i].length - 1]);

}

}

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

EquationsSet S = **new** EquationsSet();

{

**double**[][] t = {{1.43, 0.87, -1.57, -0.58, 2.3},

{0.63, -0.57, -2.34, 0.66, 0.77},

{1.57, 0.66, -0.57, 1.15, -0.2},

{0.88, -0.67, 0.55, -0.45, 0.56}};

S.Coefficients = t;

}

S.print();

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

**double**[] t = S.Solve();

System.***out***.println("Solution:");

**for** (**double** e : t) {

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

}

}

**private** **double**[][] Simplify() {

**int**[] notNull = FindFirstNotNull();

Make1LineNotStartingWithNull(notNull[0], notNull[1]);

// Всі елементи 1 рядка діляться на 1 елемент обраної колонки

DivideOn(0, Coefficients[notNull[0]][notNull[1]]);

// Отримання нулів у першій колонці

MakeNulls();

EquationsSet es = **new** EquationsSet();

es.Coefficients = GetTruncatedArray();

**if** (es.Coefficients.length > 1) { // Роблю так доки не залишається

// жодного рівняння.

// Рекурсивний виклик самої себе для обробки масиву

// коефіцієнтів розміром n-1

es.Simplify();

} **else** {

es.Coefficients[0][1] /= es.Coefficients[0][0];

es.Coefficients[0][0] = 1;

}

**this**.RIP(es);

**return** **this**.Coefficients;

}

**private** **int**[] FindFirstNotNull() {

**int**[] Result = **new** **int**[2];

Result[0] = 0;

Result[1] = 0;

find\_column: **for** (Result[0] = 0; Result[0] < Coefficients.length; Result[0]++) {

**for** (Result[1] = 0; Result[1] < Coefficients.length - 1; Result[1]++) {

**if** ((Coefficients[Result[1]][Result[0]]) != 0)

**break** find\_column;

}

}

**return** Result;

}

**private** **void** Make1LineNotStartingWithNull(**int** i, **int** h) {

**if** (Coefficients[0][0] == 0) {

// Знаходжу рядок, що не починається з нуля

**int** j;

**for** (j = 1; j < Coefficients.length; j++) {

**if** ((Coefficients[j][0]) != 0)

**break**;

**else** **if** (j == Coefficients.length - 1) {

// Немає ненульових коефіцієнтів у першій колонці

// Метод не спрацює

System.***out***

.println("Помилка: У першій колонці нема ненульових елементів...");

System.***out***.println("Коректна робота не гарантується");

// System.exit(-1); //Можна б і завершити програму

}

}

**double**[] t = **new** **double**[Coefficients[0].length];

// Запам'ятовую старі коефіцієнти першого рівняння

t = Coefficients[0];

Coefficients[0] = Coefficients[j];

Coefficients[j] = t;

}

}

**private** **void** DivideOn(**int** lineNumber, **double** p) {

**for** (**int** j = 0; j < Coefficients[0].length; j++) {

Coefficients[lineNumber][j] /= p;

}

}

**private** **void** MakeNulls() {

**for** (**int** k = 1; k < Coefficients.length; k++) { // Для кожного рівн.,

// поч з ІІ

**double** t = Coefficients[k][0]; // Перший елемент кожного рівняння

**for** (**int** l = 0; l < Coefficients[0].length; l++) {// Для кожного

// коефіцієнта

// рівняння

Coefficients[k][l] -= Coefficients[0][l] \* t;

}

}

}

**private** **double**[][] GetTruncatedArray() {

**double**[][] result = **new** **double**[Coefficients.length - 1][Coefficients[0].length - 1];

**for** (**int** k = 1; k < Coefficients.length; k++) {

**for** (**int** l = 1; l < Coefficients[0].length; l++) {

result[k - 1][l - 1] = Coefficients[k][l];

}

}

**return** result;

}

**private** **void** RIP(EquationsSet o) {

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

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

// Рядок із врахуванням зміщення у коефіцієнтах викл.

// екземплярів

**int** line = **this**.Coefficients.length - 1 - i;

// Колонка із врахуванням зміщення у коефіцієнтах

// викл.екземплярів

**int** column = **this**.Coefficients[line].length - 1 - j;

// Номер рядка у коеф. параметра

**int** parLine = o.Coefficients.length - i - 1;

// Номер колонки у коеф. параметра

**int** parCol = o.Coefficients[parLine].length - j - 1;

// Переписую коефіцієнти із зміщенням

**this**.Coefficients[line][column] = o.Coefficients[parLine][parCol];

}

}

}

**private** **void** MakeNullsInUpperLines() {

**int** j = 0;

**for** (**int** i = Coefficients.length - 1; i >= 0; i--) { // Номер

// активного

// рівняння

**for** (j = i - 1; j >= 0; j--) { // Номер рівняння, яке

// перетворюється

// Номер останнього коефіцієнта (вільного члена) у рядку

**int** t = Coefficients[i].length - 1;

// Проводжу операцію над вільним членом

Coefficients[j][t] -= Coefficients[j][i] \* Coefficients[i][t];

// Коефіцієнт над одиницею перетворюється в нуль

Coefficients[j][i] = 0;

}

}

}

}