Ministry of Education and Science of Ukraine
Kharkiv National University of Radioelectronics

Department of System Techniques

Subject:
«Optimization method and operations»

Practice work №3
«Markov chains research»

Done by
Avdan O.,
Buriak D.,
Kuzmin A.
Filonova N.,
Iatsenko A.
Group KHT-19-1

Taken by
Vyshniak M.Y.
with the mark «_____»

Kharkiv
2021

1.1 Our team:

Our team consists of 5 participants, among which: Olexandra Avdan, Buriak Daniil, Kuzmin Artem, Filonova Nadiia and Iatsenko Anna, all from the group KHT-19-1.

1.2 Purpose of work:

To explore the implementation of Markov chains, explore the theoretical and practical base concerning this theme.

1.3 Tasks for the following practice:

The following practice consists of three different tasks according to our group number.

1.4 Algorithm description:

In probability theory and related fields, a *stochastic or random process* is a mathematical object usually defined as a family of random variables. Many stochastic processes can be represented by time series. However, a stochastic process is by nature continuous while a time series is a set of observations indexed by integers.

A *Markov chain* is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. A countably infinite sequence, in which the chain moves state at discrete time steps, gives a discrete-time Markov chain. A continuous-time process is called a continuous-time Markov chain. It is named after the Russian mathematician Andrey Markov.

Markov chains have many applications as statistical models of real-world processes, such as studying cruise control systems in motor vehicles, queues or lines of customers arriving at an airport, currency exchange rates and animal population dynamics.

1.5 Pace of work

Code, that implements Tasks 1-3 written on *Java*, because to our concern this programming language is an optimal combination of convenience and usability. All the code contains comments concerning certain functions and whole classes to make a code more understandable.

1.5.1 Task 1

Task 1.

Find the states probabilities in one, two, three, four, five steps.
Draw graphs for each of the state probabilities depending on the step number.

Option:

Team2:

$$\mathbf{P} = \begin{bmatrix} 0.3 & 0.5 & 0.1 & 0.1 \\ 0.4 & 0.2 & 0.2 & 0.2 \\ 0.1 & 0.1 & 0.6 & 0.2 \\ 0.2 & 0.4 & 0.3 & 0.1 \end{bmatrix}$$

Task 1 implementation:

Class Insert

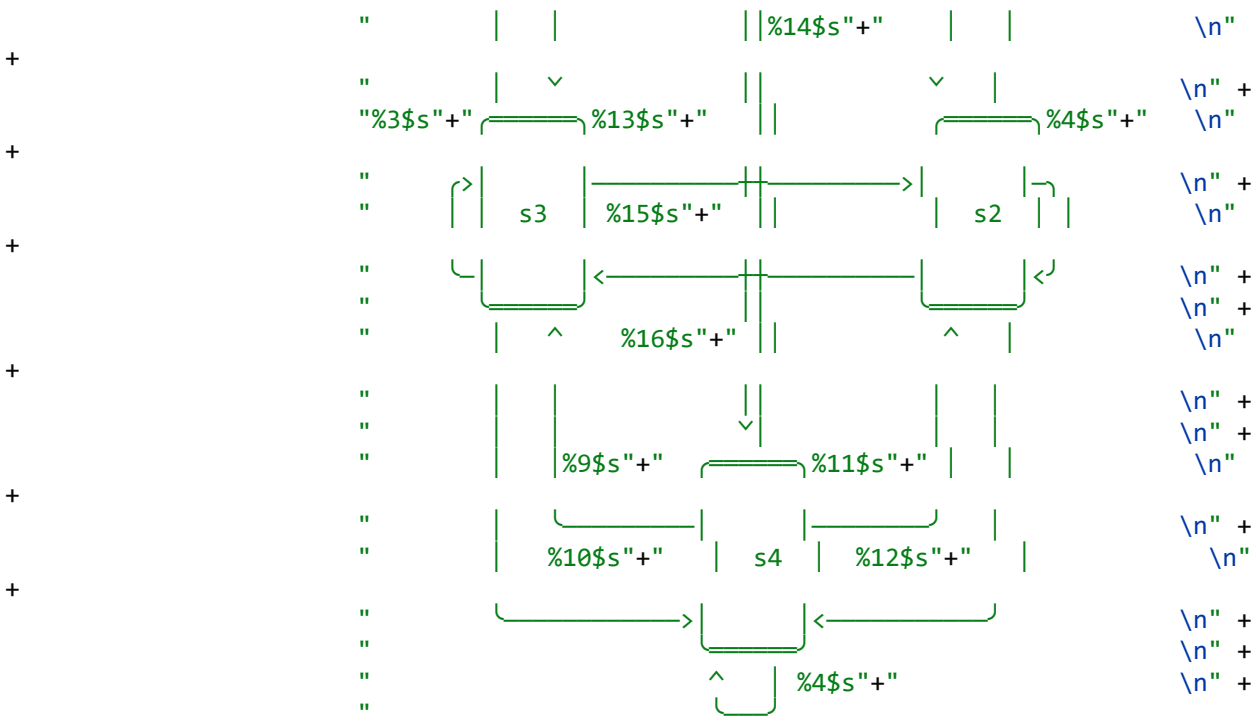
```
public class Insert {

    public static double[][] readP() {
        return new double[][]{
                {0.3,0.5,0.1,0.1},
                {0.4,0.2,0.2,0.2},
                {0.1,0.1,0.6,0.2},
                {0.2,0.4,0.3,0.1}};
    }
}
```

Class SquareMatrix

```
import java.util.Map;
import java.util.TreeMap;

public class SquareMatrix {
int x;
public double[][] p1;
public Map<Integer,double[][]> p;

    static char upLeft = '╔';
    static char upRight = '╗';
    static char upMiddle = '╦';
    static char downLeft = '╚';
    static char downRight = '╝';
    static char downMiddle = '╩';
    static char horizontal = '═';
    static char vertical ='║';
    static char center = '╬';
    static char leftCorner = '╠';
    static char rightCorner = '╣';

    public SquareMatrix(double[][] p) {
        this.p = new TreeMap<>();
        this.p.put(1,p);
        this.x=p.length;
        this.p1=p;
    }
```

```
    public double[][] calcPow(int a){
        if(this.p.isEmpty()||!this.p.containsKey(a)){
            double[][]matrix = pow(a);
            p.put(a,matrix);
        }
        return this.p.get(a);
    }

    public void getPow(int a){
        double[][] matrix = calcPow(a);
        int digits = calcDigit(matrix);
        StringBuilder table = new StringBuilder();
        StringBuilder cell = new StringBuilder();
        cell.append((String.valueOf(horizontal)).repeat(digits+2));
        for(int i = 0; i<x;i++){
            if(i==0){
                table.append(upLeft).append(cell);
            }
            else{
                table.append(upMiddle).append(cell);
            }
        }
        table.append(upRight).append("\n");

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

            for(int j = 0; j<x;j++){
                table.append(vertical);
                //%1$03.1f
                String str = String.format('%' + "1$" + "0" + digits + '.' + (digits
- 2) + "f",matrix[i][j]);
                table.append(" ").append(str).append(" ");
                if(j==x-1){
                    table.append(vertical).append(" ▓▓▓");
                }
            }
            table.append("\n");
            if(i!=x-1){
            table.append(leftCorner).append((String.valueOf(cell)+center).repeat(x-
1)).append(cell).append(rightCorner).append(" ▓▓▓").append("\n");
            }


        }

        for(int i = 0; i<x;i++){
            if(i==0){
                table.append(downLeft).append(cell);
            }
            else{
                table.append(downMiddle).append(cell);
            }
        }
        table.append(downRight).append(" ▓▓▓").append("\n").append("
▓").append("▓".repeat((digits+3)*x)).append("▓").append("\n").append("
▓").append("▓".repeat((digits+3)*x)).append("▓\n");
        System.out.println(table);
    }

    public void writeGraph(int b) {
```

```
        String str;
        double[][] a = calcPow(b);
        if (a.length==2){
            str =    ""+   "  ┌──────┐ %1$03.1f"+ "   ┌──────┐          \n" +
                     ""+   "  │      │"+  "  ──────>"+ "  │      │          \n" +
                     ""+   "  │  s1  │       "+       "  │  s2  │          \n" +
                     ""+   "  │      │"+  "  <──────"+ "  │      │          \n" +
                     ""+   "  └──────┘ %2$03.1f"+ "   └──────┘          \n";
            System.out.printf(str,a[0][1],a[1][0]);
        }
        else if (a.length==3){

            String[][] c = ToString(a);
            str =    "                                                   \n"+
                     "                                                   \n"+
                     "           %1$s"+"      ┌──────┐    %3$s"+"        \n"+
                     "      ┌──────────────>  │      │ <──────────┐      \n"+
                     "      │     %2$s"+"     │  s1  │    %4$s"+" │      \n"+
                     "      │   ┌──────────── │      │ ────────┐  │      \n"+
                     "      │   │             └──────┘         │  │      \n"+
                     "      │   │                              │  │      \n"+
                     "      │   │                              │  │      \n"+
                     "      │   v                              v  │      \n"+
                     "   ┌──────┐           %5$s"+"         ┌──────┐      \n"+
                     "   │      │ ──────────────────────>   │      │      \n"+
                     "   │  s3  │           %6$s"+"         │  s2  │      \n"+
                     "   │      │ <──────────────────────   │      │      \n"+
                     "   └──────┘                           └──────┘      \n";
            System.out.printf((str) +
"%n",c[2][0],c[0][2],c[1][0],c[0][1],c[2][1],c[1][2]);
        }
        else if (a.length==4){
            if (a[0][3]==0 && a[3][0]==0){
                str =    ""+   "  ┌──────┐ %1$03.1f"+ "   ┌──────┐          \n" +
                         ""+   "  │      │"+  "  ──────>"+ "  │      │          \n" +
                         ""+   "  │  s1  │       "+       "  │  s2  │          \n" +
                         ""+   "  │      │"+  "  <──────"+ "  │      │          \n" +
                         ""+   "  └──────┘ %2$03.1f"+ "   └──────┘          \n" +
                         "          "+          "                             \n" +
                         "  %3$03.1f │ ^        "+"  %5$03.1f │ ^          \n" +
                         ""+     "     │ │  %4$03.1f "+"       │ │  %6$03.1f\n" +
                         ""+     "     v │       "+          "  v │           \n" +
                         ""+   "  ┌──────┐ %7$03.1f"+ "   ┌──────┐          \n" +
                         ""+   "  │      │"+  "  ──────>"+ "  │      │          \n" +
                         ""+   "  │  s3  │       "+       "  │  s4  │          \n" +
                         ""+   "  │      │"+  "  <──────"+ "  │      │          \n" +
                         ""+   "  └──────┘ %8$03.1f  "+ "  └──────┘          \n";

System.out.printf(str,a[1][0],a[0][1],a[2][0],a[0][2],a[3][1],a[1][3],a[3][2],a[2][3]
);
            }
            else{
                String[][] c = ToString(a);
                str=    "                 ┌──┐ %1$s"+"                  \n" +
                        "                 │  v                           \n" +
                        "         %5$s"+"   ┌──────┐   %7$s"+"           \n" +
                        "   ┌──────────> │      │ <──────────┐           \n" +
                        "   │     %6$s"+"  │  s1  │  %8$s"+"   │           \n" +
                        "   │   ┌──────── │      │ ──────┐    │           \n" +
                        "   │   │         └──────┘       │    │           \n" +
                        "   │   │            │^          │    │           \n" +
```




```
\n\n";

System.out.printf(str,c[0][0],c[1][1],c[2][2],c[3][3],c[0][2],c[2][0],c[0][1],

c[1][0],c[2][3],c[3][2],c[1][3],c[3][1],c[1][2],c[0][3],c[2][1],c[3][0]);
            }
        }
    }

    private String[][] ToString(double[][] a) {
        String[][] b = new String[x][x];
        for(int i = 0; i<x; i++){
            for(int j = 0; j<x; j++){
                StringBuilder stringBuilder = new
StringBuilder(String.valueOf(a[i][j]));
                if(stringBuilder.length()!=7){
                    stringBuilder.append(" ".repeat(7-stringBuilder.length()));
                }
                b[i][j]= String.valueOf(stringBuilder);
            }
        }
        return b;
    }

    private int calcDigit(double[][] matrix) {
        int digits=2;
        boolean b=false;
        for(int i = 0; i<x; i++){
            if(b){
                break;
            }
            for(int j = 0;j<x;j++){
                int d = Double.toString(matrix[i][j]).length();
                if(d>digits){
                    digits=d;
                }
                if(digits>=7){
                    b=true;
```

```
                    break;
                }
            }
        }
        return digits;
    }

    private double[][] round(double[][] matrix){
        for(int i = 0; i<4; i++){
            for(int j = 0;j<4;j++){
                matrix[i][j] = Math.round(matrix[i][j]*100000)/100000d;
            }

        }
        return matrix;
    }

    private double[][] pow(int a) {
        int b=a-1;
        double[][] matrix = p.get(1);
        for(int i=2;i<a;i++){
            if(p.containsKey(i)){
                b=a-i;
                matrix=p.get(i);
            }
        }
        for(int i = 0; i < b; i++){
            matrix = round(multiplyMatrices(p.get(1),matrix));

        }
        return matrix;
    }
    private static double[][] multiplyMatrices(double[][] firstMatrix, double[][]
secondMatrix) {
        double[][] result = new double[firstMatrix.length][secondMatrix[0].length];

        for (int row = 0; row < result.length; row++) {
            for (int col = 0; col < result[row].length; col++) {
                result[row][col] = multiplyMatricesCell(firstMatrix, secondMatrix,
row, col);
            }
        }

        return result;
    }

    static double multiplyMatricesCell(double[][] firstMatrix, double[][]
secondMatrix, int row, int col) {
        double cell = 0;
        for (int i = 0; i < secondMatrix.length; i++) {
            cell += firstMatrix[row][i] * secondMatrix[i][col];
        }
        return cell;
    }
}
```

Class StateProbab

```
public class StateProbab {

    public static double[][] p;
```

```
    public static SquareMatrix matrix;
    public static void main(String[] args) {
        p=Insert.readP();
        matrix = new SquareMatrix(p);
        System.out.println("States probabilities on first step:");
        matrix.getPow(1);
        System.out.println("Graph for first step:");
        matrix.writeGraph(1);
        System.out.println("States probabilities on second step:");
        matrix.getPow(2);
        System.out.println("Graph for second step:");
        matrix.writeGraph(2);
        System.out.println("States probabilities on third step:");
        matrix.getPow(3);
        System.out.println("Graph for third step:");
        matrix.writeGraph(3);
        System.out.println("States probabilities on fourth step:");
        matrix.getPow(4);
        System.out.println("Graph for fourth step:");
        matrix.writeGraph(4);
        System.out.println("States probabilities on fifth step:");
        matrix.getPow(5);
        System.out.println("Graph for fifth step:");
        matrix.writeGraph(5);
        System.out.println("States probabilities on six step:");
        matrix.getPow(6);
        System.out.println("Graph for six step:");
        matrix.writeGraph(6);

    }
}
```

Task 1 output:

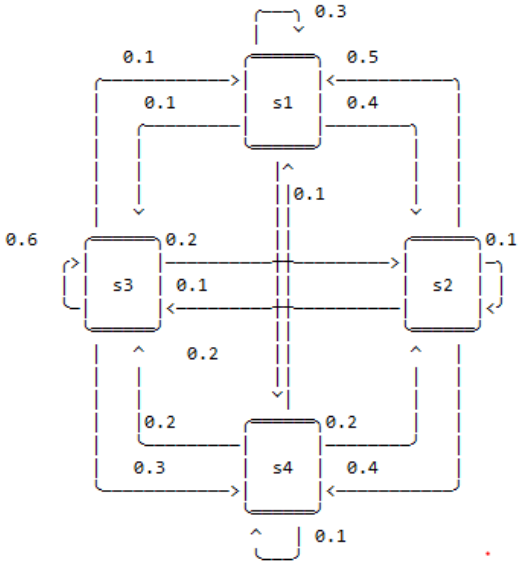
States probabilities on first step:

0,3	0,5	0,1	0,1
0,4	0,2	0,2	0,2
0,1	0,1	0,6	0,2
0,2	0,4	0,3	0,1

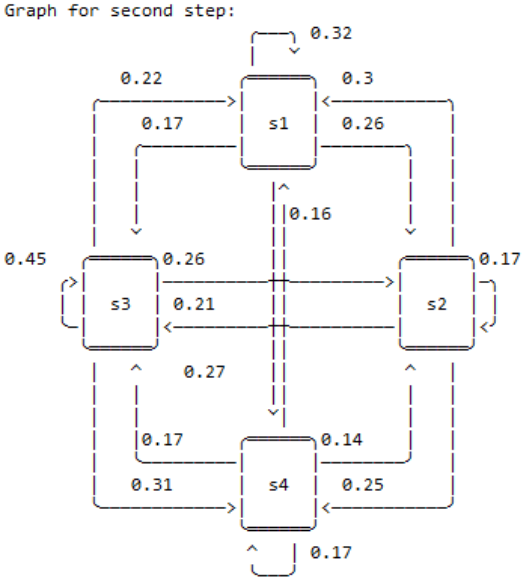
States probabilities on second step:

0,32	0,30	0,22	0,16
0,26	0,34	0,26	0,14
0,17	0,21	0,45	0,17
0,27	0,25	0,31	0,17

Graph for first step:



Graph for second step:



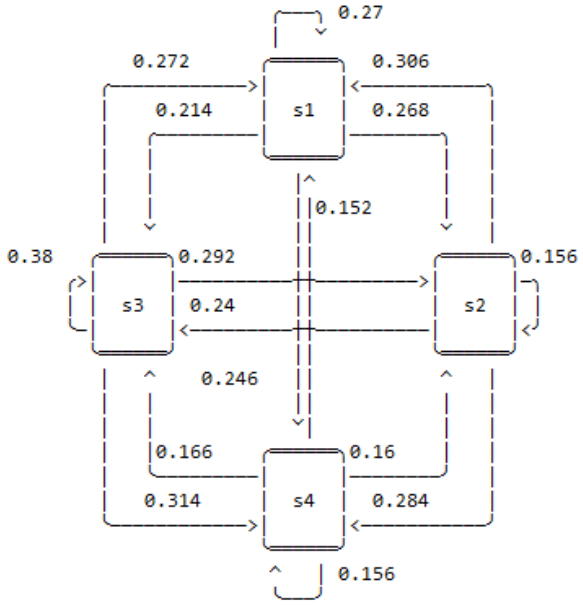
States probabilities on third step:

0,270	0,306	0,272	0,152
0,268	0,280	0,292	0,160
0,214	0,240	0,380	0,166
0,246	0,284	0,314	0,156

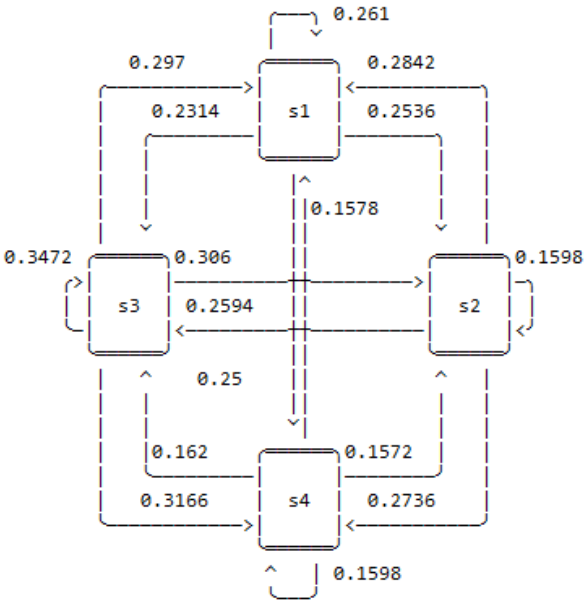
States probabilities on fourth step:

0,2610	0,2842	0,2970	0,1578
0,2536	0,2832	0,3060	0,1572
0,2314	0,2594	0,3472	0,1620
0,2500	0,2736	0,3166	0,1598

Graph for third step:



Graph for fourth step:



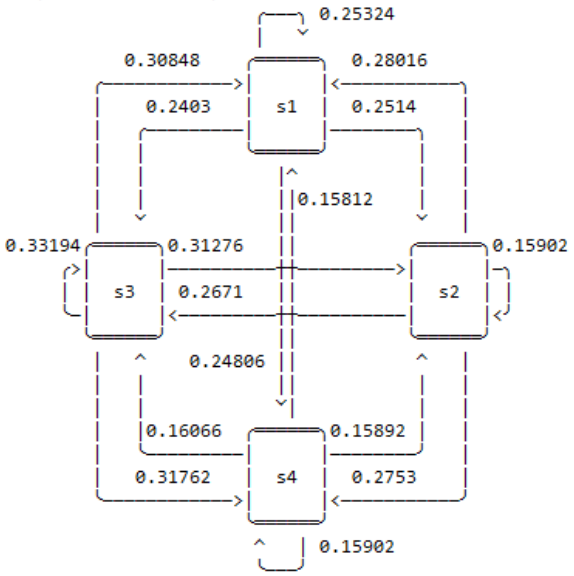
States probabilities on fifth step:

0,25324	0,28016	0,30848	0,15812
0,25140	0,27692	0,31276	0,15892
0,24030	0,26710	0,33194	0,16066
0,24806	0,27530	0,31762	0,15902

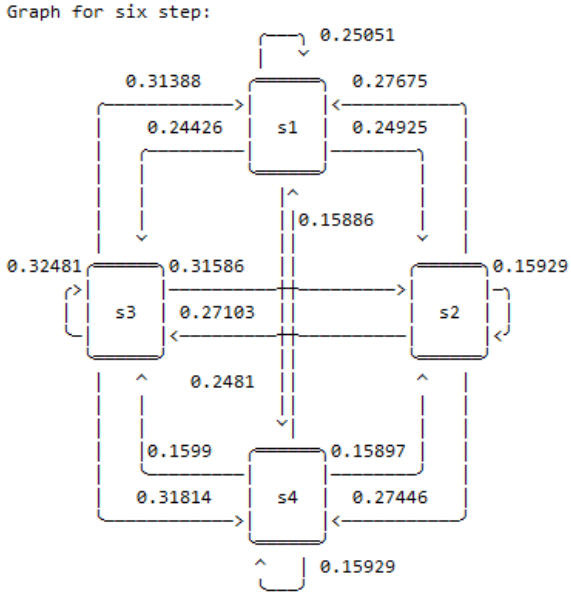
States probabilities on six step:

0,25051	0,27675	0,31388	0,15886
0,24925	0,27593	0,31586	0,15897
0,24426	0,27103	0,32481	0,15990
0,24810	0,27446	0,31814	0,15929

Graph for fifth step:



Graph for six step:

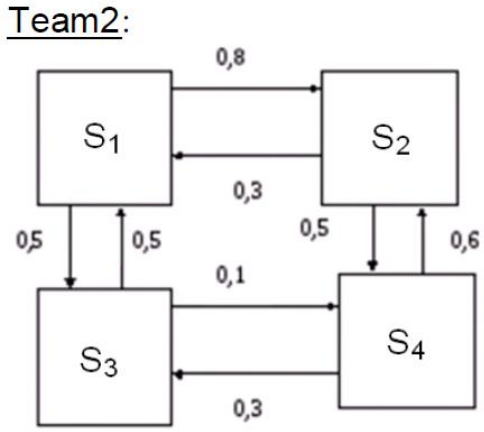


1.5.2 Task 2

Task 2.

1) Using the marked graph of the system states, compose the system of Kolmogorov equations for the limiting probabilities.
2) Find the limiting probabilities.

Option:



Task 2 implementation:

Class Insert

```
public class Insert {


    public static double[][] readA() {
        //double[][] doubles = {{0.3, 0.5}, {0.8, 0.6}, {0.5, 0.3}, {0.5, 0.1}};
        return new double[][]{
                {0,0.8,0.5,0},
                {0.3,0,0,0.5},
                {0.5,0,0,0.1},
                {0,0.6,0.3,0}
        };
    }

    public static double[] readP() {
        return new double[]{1d, 0d, 0d, 0d};
    }
}
```

Class Kolmogorov

```
public class Kolgomorov {

    public static double[][] a;
    public static double[] p;
    public static double[][] A;
    public static void main(String[] args) {
        a=Insert.readA();
        p=Insert.readP();
        Util.writeGraph(a);
        Util.calcEquatin(a);
```

```
        A=Util.convertToMatrix(a);
        p=SolveLinearEquation.solve(A);
    }
}
```

Class SolveLinearEquation

```
import java.text.DecimalFormat;

import static java.lang.Math.round;

public class SolveLinearEquation {

    static DecimalFormat f1 = new DecimalFormat("#0.0");

    public static double[] solve(double[][] matrix) {
        String[] var = {"p1", "p2", "p3", "p4"};
        // the number of variables in the equations
        int n = matrix.length;
        // the coefficients of each variable for each equations
        double[][] mat = new double[n][n];
        double[] constants = new double[n];
        ///initialization
        for (int i = 0; i < n; i++) {
            System.arraycopy(matrix[i], 0, mat[i], 0, n);
            constants[i] = -matrix[i][n];
        }
        //Matrix representation


        System.out.println("Kolmogorov matrix:");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {

                System.out.printf("%+04.1f * %s
",(double)round(mat[i][j]*10)/10,var[j]);
            }
            if(round(constants[i])==0){
                constants[i] = 0;
            }
            System.out.println(" = " + f1.format(constants[i]));
        }

        System.out.println();

        //inverse of matrix mat[][]
        double[][] inverted_mat = invert(mat);
        System.out.println("The inverse matrix is: ");
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                System.out.printf("%+05.3f  ", inverted_mat[i][j]);
            }
            System.out.println();
        }

        //Multiplication of mat inverse and constants
        double[] result = new double[n];
        for (int i = 0; i < n; i++) {
            for (int k = 0; k < n; k++) {
                result[i] = result[i] + inverted_mat[i][k] * constants[k];
```

```
            }
        }
        test(matrix, result);
        System.out.println();
        System.out.println("The product is:");
        for (int i = 0; i < n; i++) {
            System.out.printf("%s = %.3f \n", var[i], result[i]);
        }

        return result;
    }

    public static double[][] invert(double[][] a) {
        int n = a.length;
        double[][] x = new double[n][n];
        double[][] b = new double[n][n];
        int[] index = new int[n];
        for (int i = 0; i < n; ++i)
            b[i][i] = 1;
        // Transform the matrix into an upper triangle
        gaussian(a, index);
        // Update the matrix b[i][j] with the ratios stored
        for (int i = 0; i < n - 1; ++i)
            for (int j = i + 1; j < n; ++j)
                for (int k = 0; k < n; ++k)
                    b[index[j]][k]
                            -= a[index[j]][i] * b[index[i]][k];
        // Perform backward substitutions
        for (int i = 0; i < n; ++i) {
            x[n - 1][i] = b[index[n - 1]][i] / a[index[n - 1]][n - 1];
            for (int j = n - 2; j >= 0; --j) {
                x[j][i] = b[index[j]][i];
                for (int k = j + 1; k < n; ++k) {
                    x[j][i] -= a[index[j]][k] * x[k][i];
                }
                x[j][i] /= a[index[j]][j];
            }
        }
        return x;
    }

    // Method to carry out the partial-pivoting Gaussian
    // elimination.  Here index[] stores pivoting order.
    public static void gaussian(double[][] a, int[] index) {
        int n = index.length;
        double[] c = new double[n];
        // Initialize the index
        for (int i = 0; i < n; ++i)
            index[i] = i;
        // Find the rescaling factors, one from each row
        for (int i = 0; i < n; ++i) {
            double c1 = 0;
            for (int j = 0; j < n; ++j) {
                double c0 = Math.abs(a[i][j]);
                if (c0 > c1) c1 = c0;
            }
            c[i] = c1;
        }
        // Search the pivoting element from each column
        int k = 0;
        for (int j = 0; j < n - 1; ++j) {
```

```
            double pi1 = 0;
            for (int i = j; i < n; ++i) {
                double pi0 = Math.abs(a[index[i]][j]);
                pi0 /= c[index[i]];
                if (pi0 > pi1) {
                    pi1 = pi0;
                    k = i;
                }
            }

            // Interchange rows according to the pivoting order
            int itmp = index[j];
            index[j] = index[k];
            index[k] = itmp;
            for (int i = j + 1; i < n; ++i) {
                double pj = a[index[i]][j] / a[index[j]][j];

                // Record pivoting ratios below the diagonal
                a[index[i]][j] = pj;

                // Modify other elements accordingly
                for (int l = j + 1; l < n; ++l)
                    a[index[i]][l] -= pj * a[index[j]][l];
            }
        }
    }

    public static void test(double[][] mat, double[] res){
        // the number of variables in the equations
        int n = mat.length;
        // the coefficients of each variable for each equations
        double[][] matrix = new double[n][n];
        double[] constants = new double[n];
        //initialization
        for (int i = 0; i < n; i++) {
            System.arraycopy(mat[i], 0, matrix[i], 0, n);
            constants[i] = -mat[i][n];
        }
        System.out.println();
        System.out.println("Equality check A*x=b:");
        double right;
        for(int i = 0; i < n; i++){
            right = 0;
            for(int j = 0; j < n; j++){
                right += matrix[i][j]*res[j];
            }

            if(round(right)==0){
                right = 0;
            }
            if(round(constants[i])==0){
                constants[i] = 0;
            }

            if(round(right*100)/100==round(constants[i]*100)/100){
                System.out.println("Values in the string #" + (i+1) + " are equal: "
+ f1.format(right) + " = " + f1.format(constants[i]));
            }
            else{
                System.out.println("Values in the string #" + (i+1) + " aren't equal:
" + right + " != " + constants[i]);
```

```
            }
        }
    }
}
```

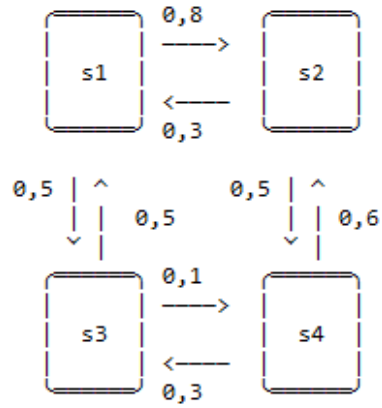
Class Util

```
import java.util.Arrays;

import static java.lang.Math.round;

public class Util {
    public static double[][] convertToMatrix(double[][] a) {
        double[][] A = new double[4][5];
        {
            A[0][0]=1;
            A[0][1]=1;
            A[0][2]=1;
            A[0][3]=1;
        }
        {
            A[1][0]=a[0][1];
            A[1][1]=-(a[1][0]+a[1][2]+a[1][3]);
            A[1][2]=a[2][1];
            A[1][3]=a[3][1];
        }
        {
            A[2][0]=a[0][2];
            A[2][1]=a[1][2];
            A[2][2]=-(a[2][0]+a[2][1]+a[2][3]);
            A[2][3]=a[3][2];
        }
        {
            A[3][0]=a[0][3];
            A[3][1]=a[1][3];
            A[3][2]=a[2][3];
            A[3][3]=-(a[3][0]+a[3][1]+a[3][2]);
        }
        {
            A[0][4]=-1;
            A[1][4]=0;
            A[2][4]=0;
            A[3][4]=0;
        }
        return A;
    }

    public static void writeGraph(double[][] a) {
        String str;
        if (a.length==2){
            str =   ""+  "      ┌──────┐  %1$03.1f"+ "      ┌──────┐       \n" +
                    ""+  "      │      │"+  " ———>"+ "      │      │       \n" +
                    ""+  "      │  s1  │     "+      "      │  s2  │       \n" +
                    ""+  "      │      │"+  " <———"+ "      │      │       \n" +
                    ""+  "      └──────┘  %2$03.1f"+ "      └──────┘       \n";
            System.out.printf(str,a[0][1],a[1][0]);
        }
        else if (a.length==4){
            if (a[0][3]==0 && a[3][0]==0){
               str =   ""+  "      ┌──────┐  %1$03.1f"+ "      ┌──────┐       \n" +
```

```
""+  "     ┌──────┐"+  " ———>"+ "  ┌──────┐        \n" +
""+  "     │  s1  │      "+       "  │  s2  │        \n" +
""+  "     └──────┘"+  " <———"+ "  └──────┘        \n" +
""+  "      ═══════  %2$03.1f"+ "   ═══════        \n" +
"         "+                "                     \n" +
  "  %3$03.1f  │ ^       "+  %5$03.1f  │ ^         \n" +
""+    "       │ │  %4$03.1f "+"       │ │ %6$03.1f\n" +
""+    "       ˅ │         "+     "     ˅ │         \n" +
""+  "      ═══════  %7$03.1f"+ "   ═══════        \n" +
""+  "     ┌──────┐"+  " ———>"+ "  ┌──────┐        \n" +
""+  "     │  s3  │      "+       "  │  s4  │        \n" +
""+  "     └──────┘"+  " <———"+ "  └──────┘        \n" +
""+  "      ═══════  %8$03.1f  "+ "   ═══════        \n";
```

```
System.out.printf(str,a[0][1],a[1][0],a[0][2],a[2][0],a[1][3],a[3][1],a[2][3],a[3][2]
);
            }
        }
    }

    public static void calcEquatin(double[][] a) {
        double[][]mat=new double[][]{{-
(a[0][1]+a[0][2]+a[0][3]),a[1][0],a[2][0],a[3][0]},{a[0][1],-
(a[1][0]+a[1][2]+a[1][3]),a[2][1],a[3][1]},{a[0][2],a[1][2],-
(a[2][0]+a[2][1]+a[2][3]),a[3][2]},{a[0][3],a[1][3],a[2][3],-
(a[3][0]+a[3][1]+a[3][2])}};
        writeEquation(mat);
    }
    public static void writeEquation(double[][] mat){
        String[] var = {"p1", "p2", "p3", "p4"};
        int n = mat.length;
        System.out.println("Kolmogorov equations:");
        for (int i = 0; i < n; i++) {
            System.out.print("p`"+(i+1)+" = ");
            for (int j = 0; j < n; j++) {

                System.out.printf("%+04.1f * %s
",(double)round(mat[i][j]*10)/10,var[j]);
            }
            System.out.println();
        }
    }
    public static double[][] normal(double[][] p) {
        double[][] p1 = new double[p.length][p.length+1];
        {
           for(int i = 0; i<p.length;i++){
               for( int j = 0; j<p.length;j++){
                   p1[i][j]=p[i][j];
               }
           }
           p1[0][3]=0;
           p1[1][3]=0;
           p1[2]=new double[]{1,1,1,-1};
        }
        return p1;
    }
}
```

Task 2 output:




```
Kolmogorov equations:
p`1 = -1,3 * p1 +0,3 * p2 +0,5 * p3 +0,0 * p4
p`2 = +0,8 * p1 -0,8 * p2 +0,0 * p3 +0,6 * p4
p`3 = +0,5 * p1 +0,0 * p2 -0,6 * p3 +0,3 * p4
p`4 = +0,0 * p1 +0,5 * p2 +0,1 * p3 -0,9 * p4
Kolmogorov matrix:
+1,0 * p1 +1,0 * p2 +1,0 * p3 +1,0 * p4  = 1,0
+0,8 * p1 -0,8 * p2 +0,0 * p3 +0,6 * p4  = 0,0
+0,5 * p1 +0,0 * p2 -0,6 * p3 +0,3 * p4  = 0,0
+0,0 * p1 +0,5 * p2 +0,1 * p3 -0,9 * p4  = 0,0

The inverse matrix is:
+0,179  +0,752  +0,439  +0,846
+0,343  -0,792  +0,580  +0,047
+0,259  +0,431  -1,207  +0,172
+0,219  -0,392  +0,188  -1,066

Equality check A*x=b:
Values in the string #1 are equal: 1,0 = 1,0
Values in the string #2 are equal: 0,0 = 0,0
Values in the string #3 are equal: 0,0 = 0,0
Values in the string #4 are equal: 0,0 = 0,0

The product is:
p1 = 0,179
p2 = 0,343
p3 = 0,259
p4 = 0,219
```

1.5.3 Task 3

Task 3.

The matrix of transitions intensities (rates) of a continuous Markov chain is given.
1) Create a marked system state graph corresponding to this matrix.
2) Compose the system of Kolmogorov equations for the limiting probabilities.
3) Find the limiting probabilities.

Option:

Team2:

$$\Lambda = \begin{pmatrix} -4 & 2 & 2 \\ 0 & -3 & 3 \\ 1 & 1 & -2 \end{pmatrix}$$

Task 3 implementation:

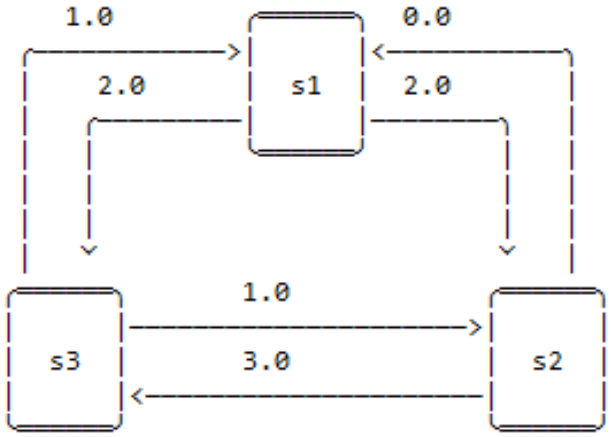
Class Insert

```
public class Insert {

    public static double[][] readP() {
        return new double[][]{
                {-4,2,2},
                {0,-3,3},
                {1,1,-2}
        };
    }
}
```

Class TransitionIntensities

```
public class TransitionIntensities {
    public static double[][] p;
    public static SquareMatrix matrix;
    public static double[] results;
    public static void main(String[] args) {
        p = Insert.readP();
        matrix = new SquareMatrix(p);
        matrix.writeGraph(1);
        Util.writeEquation(p);
        p = Util.normal(p);
        results = SolveLinearEquation.solve(p);
    }
}
```

Task 3 output:




```
Kolmogorov equations:
p`1 = -4,0 * p1 +2,0 * p2 +2,0 * p3
p`2 = +0,0 * p1 -3,0 * p2 +3,0 * p3
p`3 = +1,0 * p1 +1,0 * p2 -2,0 * p3
Kolmogorov matrix:
-4,0 * p1 +2,0 * p2 +2,0 * p3  = 0,0
+0,0 * p1 -3,0 * p2 +3,0 * p3  = 0,0
+1,0 * p1 +1,0 * p2 +1,0 * p3  = 1,0

The inverse matrix is:
-0,167  -0,000  +0,333
+0,083  -0,167  +0,333
+0,083  +0,167  +0,333

Equality check A*x=b:
Values in the string #1 are equal: 0,0 = 0,0
Values in the string #2 are equal: 0,0 = 0,0
Values in the string #3 are equal: 1,0 = 1,0

The product is:
p1 = 0,333
p2 = 0,333
p3 = 0,333
```

1.6 Conclusion

During this practice the implementation of Markov chains was explored. Also we explored the theoretical and practical base concerning this theme. Three tasks have been implemented. All the necessary code and methods output are included in the report.