Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 2

Дисциплина: Моделирование

Тема: Марковские процессы. Уравнения Колмогорова

Студент: Платонова О. С.

Группа: ИУ7-75Б

Оценка (баллы): _____

Преподаватель: Рудаков И. В.

Москва, 2021 г.

Цель работы: определение времени пребывания сложной системы в каждом из состояний.

Марковские процессы

Случайный процесс в системе S называется *марковским*, если он обладает следующим свойством: для каждого момента времени t_0 вероятность любого состояния системы в будущем зависит только от ее состояния в настоящем и не зависит от того, когда и каким образом система пришла в это состояние.

Для марковского процесса составляют уравнение Колмогорова: $F(P^I(t), P(t), \lambda) = 0$. Для решения поставленной задачи требуется составить систему уравнений. Каждое из уравнений формируется, согласно следующему правилу: левая часть содержит производную вероятности состояния; правая часть – количество членов, равное числу стрелок состояния. Причем элемент имеет знак «+», если стрелка направлена в состояние, и «–», если из состояния.

Далее будет рассмотрено решение задачи на примере системы с 3 состояниями (рисунок 1).

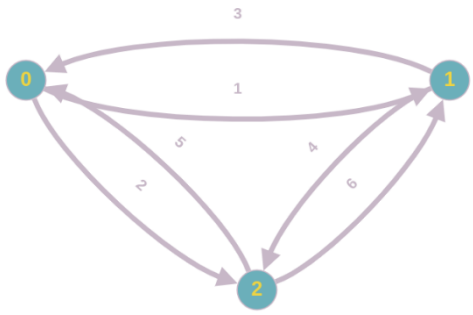


Рисунок 1.

Система уравнений Колмогорова

$$\begin{cases} P_0^I(t) = \lambda_{10}P_1(t) + \lambda_{20}P_2(t) - \lambda_{01}P_1(t) - \lambda_{02}P_1(t) \\ P_1^I(t) = \lambda_{01}P_0(t) + \lambda_{21}P_2(t) - \lambda_{10}P_1(t) - \lambda_{12}P_1(t) \\ P_2^I(t) = \lambda_{02}P_0(t) + \lambda_{12}P_1(t) - \lambda_{20}P_2(t) - \lambda_{21}P_2(t) \end{cases}$$

Для нахождения предельных вероятностей следует приравнять левые части к 0 и добавить условие нормировки к полученной системе. В результате решения полученной системы уравнений методом Гаусса, будут получены исходные вероятности.

Для определения времени пребывания системы в каждом из состояний, следует итерационно рассчитать вероятность от времени t + Δt. В момент, когда разница между найденной вероятностью и финальной не будет превышать заданной *eps* погрешности, итерацию следует завершить.

Результаты

Матрица интенсивностей

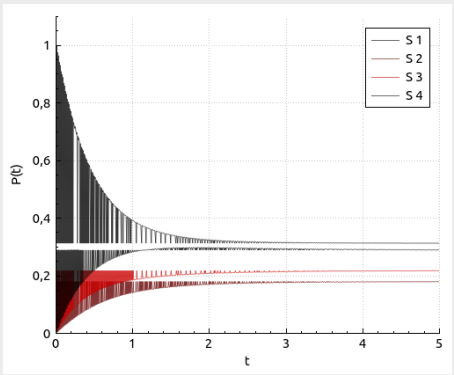
Число состояний: 4

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

Рассчитать

Результат

	Вероятность	Время
1	0,312796	1,011
2	0,180095	1,133
3	0,218009	1,143
4	0,289099	0,281



Матрица интенсивностей

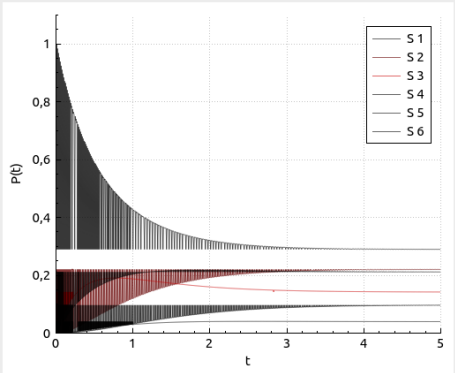
Число состояний: 6

	1	2	3	4	5	6
1	0	0	5	3	0	0
2	1	0	0	2	0	3
3	9	6	0	2	0	0
4	0	0	4	0	3	1
5	6	7	3	0	0	0
6	6	2	0	1	0	0

Рассчитать

Результат

	Вероятность	Время
1	0,289666	0,696
2	0,220473	0,714
3	0,141917	0,708
4	0,211344	0,195
5	0,039627	0,279
6	0,0969736	0,791



Листинг

Файл calculations.cpp

```cpp
const static double eps = 1e-3;

vector<double> dp(float **system, vector<double> Pcur, const int n)
{
    vector<double> DPcur(n, 0);

    for (int i = 0; i < n; i++) {
        double s = 0;
        for (int j = 0; j < n; j++) {
            s += Pcur[j] * system[i][j];
        }
        DPcur[i] = s * eps;
    }

    return DPcur;
}

void getDeltas(vector<double> &dP, vector<double> &dTimes, float **system, const int numStates)
{
    vector<double> Pcur(numStates, 0);
    Pcur[0] = 1;

    double curTime = 0;

    while (curTime < 5) {
        dP.insert(dP.end(), Pcur.begin(), Pcur.end());
        vector<double> DPcur = dp(system, Pcur, numStates);
        for (int i = 0; i < numStates; i++) {
            Pcur[i] += DPcur[i];
        }
        curTime += eps;

        dTimes.push_back(curTime);

    }
}
```

```cpp
vector<double> getTimeSystem(float **system, vector<double> P, const int numStates)
{
    vector<double> Pcur(numStates, 0);
    Pcur[0] = 1;

    double curTime = 0;
    vector<double> times(numStates, 0);

    while(find(times.begin(), times.end(), 0.0) != times.end()) {
        vector<double> DPcur = dp(system, Pcur, numStates);

        for (int i = 0; i < numStates; i++) {
            if (abs(Pcur[i] - P[i]) <= eps and
                DPcur[i] <= eps and
                times[i] == 0) {
                 times[i] = curTime;
            }
            Pcur[i] += DPcur[i];
        }
        curTime += eps;
    }

    return times;
}
```

```
Pt calculateTimeSystem(float **matrix, const int numStates)
{
    vector<float> sumIntensity(numStates, 0);
    for (int i = 0; i < numStates; i++) {
        for (int j = 0; j < numStates; j++) {
            sumIntensity[i] += matrix[i][j];
        }
    }

    float **system = nullptr;
    allocateMatrix(&system, numStates);

    for (int i = 0; i < numStates; i++) {
        for (int j = 0; j < numStates; j++) {
            if (i == j) {
                system[i][j] = -sumIntensity[i];
            }
            else {
                system[i][j] = matrix[j][i];
            }
        }
    }

    //system normalization
    float **systemNorm = nullptr;
    allocateMatrix(&systemNorm, numStates);
    copyMatrix(&systemNorm, system, numStates);

    for (int j = 0; j < numStates; j++) {
        systemNorm[numStates - 1][j] = 1;
    }
    vector<float> states(numStates - 1, 0);
    states.push_back(1);

    //system solution
    vector<double> x = gauss(systemNorm, states, numStates);
    freeMatrix(&systemNorm, numStates);
```

```

    //time calculation
    vector<double> times = getTimeSystem(system, x, numStates);

    //delta calculation
    vector<double> dP, dTimes;
    getDeltas(dP, dTimes, system, numStates);

    freeMatrix(&system, numStates);

    Pt pt;
    pt.P = x; pt.Time = times;
    pt.dP = dP; pt.dTime = dTimes;

    return pt;
}

```