



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 _____ Информационные технологии и системы управления
КАФЕДРА _____ Программное обеспечение ЭВМ и информационные технологии

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ ПО КУРСУ «ПЛАНИРОВАНИЕ ЭКСПЕРИМЕНТОВ»

Лабораторная работа №2

Студент _____
ИУ7-83Б
(Группа)

_____ Зыкин Д.А.
(И.О.Фамилия)

Преподаватель

(Подпись, дата)

_____ Куров А.В.
(И.О.Фамилия)

2020 г.

Цель работы

Реализация полного факторного эксперимента на имитационной модели функционирования СМО для нахождения среднего времени ожидания заявок в очереди.

Условие работы

1. составить матрицу планирования для проведения ПФЭ для одноканальной СМО с одним генератором заявок;
2. получить зависимость выходной величины от загрузки;
3. по результатам ПФЭ вычислить коэффициенты линейной и частично нелинейной регрессионной зависимости;
4. предусмотреть возможность сравнения рассчитанной величины с реальной, полученной по результатам имитационного моделирования.

Теоретическая часть

В данной лабораторной работе интервалы для факторов подбираются таким образом, чтобы загрузка системы лежала в промежутке от 0 до 1. Таким образом обеспечивается функционирование системы в стационарном режиме.

Факторами в данном эксперименте будут являться интенсивности генератора и обработчика, а также коэффициент дисперсии. По условию первой лабораторной работы генератор заявок функционирует по закону Рэлея, а обработчик по нормальному закону.

Для генерации времени обработки на основе заданных промежутков рассчитываются параметры распределений. Для генератора будет определяться сигма по формуле:

$$\sigma = \frac{1}{\sqrt{\frac{\pi}{2} I}} \quad (1)$$

Для обработчика параметры распределения будут определяться по формуле:

Первым фактором эксперимента будет являться интенсивность генератора, вторым - интенсивность обработчика, третьим - коэффициент дисперсии.

В ходе ПФЭ будут получено 8 коэффициентов:

- b_0 – перед свободным членом;
- b_1 – перед интенсивностью генератора;
- b_2 – перед интенсивностью обработчика;
- b_3 – перед коэффициентом дисперсии;
- остальные – перед комбинациями факторов.

Интерфейс программы

The screenshot shows a software window titled 'MainWindow' with a dark header bar. The interface is divided into several sections:

- Parameters Section:** Contains input fields for 'Генератор' (Generator) with a 'Сигма' (Sigma) value of 10,000 and 'Обработчик' (Processor) with an 'Интенсивность' (Intensity) of 0,306. There is also a 'Дисперсия' (Dispersion) field with a value of 0,314 and a 'Время моделирования' (Modeling time) field with a value of 10000. A 'Моделировать' (Model) button is present.
- Experiment Section:** Contains a table of experimental parameters and results. The table has columns for 'Интенсивность генератора минимальная', 'Интенсивность генератора максимальная', 'Интенсивность обработчика минимальная', 'Интенсивность обработчика максимальная', and 'Дисперсия минимальная'. The values are: 0,190, 0,200, 0,305, 0,309, 0,300, 0,315.
- Results Section:** Contains a list of experimental results for the 'Заполнение матрицы' (Matrix filling) experiment. It lists parameters like 'Интенсивность генератора', 'Интенсивность обработчика', 'Дисперсия', 'Ожидаемая нагрузка' (Expected load), and 'Полученная нагрузка' (Received load) for various experiments.
- Coefficients Section:** Contains two lists of coefficients. The first list is for the 'Линейного плана' (Linear plan) and the second is for the 'Частично нелинейного плана' (Partially non-linear plan). The coefficients are: $b_0 = 0.96$, $b_1 = 0.37$, $b_2 = -0.12$, $b_3 = 0.14$, $b_{12} = -0.03$, $b_{13} = 0.05$, $b_{23} = -0.05$, $b_{123} = -0.04$.
- Summary Section:** Contains a table of results for the 'Линейного плана' (Linear plan) and 'Частично нелинейного плана' (Partially non-linear plan). The results are: $b_0 = 0.96$, $b_1 = 0.37$, $b_2 = -0.12$, $b_3 = 0.14$, $b_{12} = -0.03$, $b_{13} = 0.05$, $b_{23} = -0.05$, $b_{123} = -0.04$.

Для проведения ПФЭ пользователь должен задать три интервала. Они задаются в строках, расположенных после кнопки *Эксперимент*.

Также после проведения ПФЭ полученные коэффициенты используются для расчета отклонения от результатов моделирования. Пользователь может задать параметры для моделирования, которые будут использоваться при проверке, в окнах, расположенных над кнопкой *Эксперимент*.

При проведении моделирования для заполнения столбца u , пользователю выводятся расчетные и реальные загрузки для каждой серии. После чего выводятся полученные коэффициенты для линейного и частично нелинейного планов.

На скриншоте выше приведен пример работы программы для интервала загрузки системы [0.624;0.662].

Из графика, полученного в первой лабораторной работы, следует, что линейный план должен иметь большую точность, так как график на данном отрезке линеен. Для аппроксимации линейной функции используется линейная функция.

Реализованные классы и методы

Для проведения ПФЭ были разработаны следующие классы:

- FExperiment
- PlotGenerator

FIntenseExperiment

Данный класс предназначен для хранения информации об эксперименте (интервалах, матрице эксперимента, контрольных значениях и т.д.)

```
1 public class FIntenseExperiment
2 {
3     int paramsAmount = 3;
4     int coeffsAmount = 8;
5
6     int[,] matrix =
7     {
8         {1, -1, -1, -1, 1, 1, 1, -1},
9         {1, -1, -1, 1, 1, -1, -1, 1},
10        {1, -1, 1, -1, -1, 1, -1, 1},
11        {1, -1, 1, 1, -1, -1, 1, -1},
12        {1, 1, -1, -1, -1, -1, 1, 1},
13        {1, 1, -1, 1, -1, 1, -1, -1},
14        {1, 1, 1, -1, 1, -1, -1, -1},
15        {1, 1, 1, 1, 1, 1, 1, 1},
16    };
17
18    double[] genInterval;
19    double[] procInterval;
20    double[] procSigmaInterval;
21
22    double genIntense;
23    double procIntense;
24    double procSigma;
25
26    public double[] ExperimentResults { get; set; }
```

```

27         double[] coeffs;
28         double[] x;
29
30         double[] zCenterList;
31         double[] zDeltaList;
32
33         public FIntenseExperiment(double minGenIntense,
double maxGenIntense, double minProcIntense, double
maxProcIntense,
34         double minProcSigma, double maxProcSigma, double
genIntense, double procIntense, double procSigma)
35     {
36         genInterval = new double[2];
37         procInterval = new double[2];
38         procSigmaInterval = new double[2];
39
40         ExperimentResults = new double[coeffsAmount];
41         coeffs = new double[coeffsAmount];
42
43         zDeltaList = new double[paramsAmount];
44         zCenterList = new double[paramsAmount];
45
46         x = new double[paramsAmount];
47         coeffs = new double[coeffsAmount];
48
49         genInterval[0] = minGenIntense; genInterval[1] =
maxGenIntense;
50         procInterval[0] = minProcIntense;
procInterval[1] = maxProcIntense;
51         procSigmaInterval[0] = minProcSigma;
procSigmaInterval[1] = maxProcSigma;
52
53         this.genIntense = genIntense;
54         this.procIntense = procIntense;
55         this.procSigma = procSigma;
56     }
57
58     public List<string> GetModellingResults()
59     {
60         List<string> results = new List<string>();
61         ExperimentResults[0] =
PlotGenerator.IntenseExperiment(genInterval[0],
procInterval[0], procSigmaInterval[0], out string res);
62         results.Add(res);

```

```

63         ExperimentResults[1] =
        PlotGenerator.IntenseExperiment(genInterval[0],
        procInterval[0], procSigmaInterval[1], out res);
64         results.Add(res);
65         ExperimentResults[2] =
        PlotGenerator.IntenseExperiment(genInterval[0],
        procInterval[1], procSigmaInterval[0], out res);
66         results.Add(res);
67         ExperimentResults[3] =
        PlotGenerator.IntenseExperiment(genInterval[0],
        procInterval[1], procSigmaInterval[1], out res);
68         results.Add(res);
69         ExperimentResults[4] =
        PlotGenerator.IntenseExperiment(genInterval[1],
        procInterval[0], procSigmaInterval[0], out res);
70         results.Add(res);
71         ExperimentResults[5] =
        PlotGenerator.IntenseExperiment(genInterval[1],
        procInterval[0], procSigmaInterval[1], out res);
72         results.Add(res);
73         ExperimentResults[6] =
        PlotGenerator.IntenseExperiment(genInterval[1],
        procInterval[1], procSigmaInterval[0], out res);
74         results.Add(res);
75         ExperimentResults[7] =
        PlotGenerator.IntenseExperiment(genInterval[1],
        procInterval[1], procSigmaInterval[1], out res);
76         results.Add(res);
77
78         return results;
79     }
80
81     public double[] GetCoeffs()
82     {
83         for (int j = 0; j < coeffsAmount; j++)
84         {
85             double sum = 0;
86             for (int i = 0; i < coeffsAmount; i++)
87             {
88                 sum += matrix[i, j] *
ExperimentResults[i];
89             }
90             coeffs[j] = sum / coeffsAmount;
91         }
92

```

```

93         return coeffs;
94     }
95
96     public double[] GetExperimentParams()
97     {
98         zCenterList[0] = (genInterval[1] +
99 genInterval[0]) / 2;
100         zDeltaList[0] = (genInterval[1] -
101 genInterval[0]) / 2;
102
103         zCenterList[1] = (procInterval[1] +
104 procInterval[0]) / 2;
105         zDeltaList[1] = (procInterval[1] -
106 procInterval[0]) / 2;
107
108         zCenterList[2] = (procSigmaInterval[1] +
109 procSigmaInterval[0]) / 2;
110         zDeltaList[2] = (procSigmaInterval[1] -
111 procSigmaInterval[0]) / 2;
112
113         x[0] = (genIntense - zCenterList[0]) /
114 zDeltaList[0];
115         x[1] = (procIntense - zCenterList[1]) /
116 zDeltaList[1];
117         x[2] = (procSigma - zCenterList[2]) /
118 zDeltaList[2];
119
120         return x;
121     }
122
123     public double LinExperiment()
124     {
125         return 1 * coeffs[0] + (x[0] * coeffs[1]) +
126 (x[1] * coeffs[2]) + (x[2] * coeffs[3]);
127     }
128
129     public double NonLinExperiment()
130     {
131         return 1 * coeffs[0] + (x[0] * coeffs[1]) +
132 (x[1] * coeffs[2]) + (x[2] * coeffs[3]) +
133 (x[0] * x[1] * coeffs[4]) + (x[0] * x[2] *
134 coeffs[5]) + (x[1] * x[2] * coeffs[6]) +
135 (x[0] * x[1] * x[2] * coeffs[7]);
136     }
137 }

```


PlotGenerator

Данный класс предназначен для определения среднего времени ожидания заявки в очереди после проведении серии экспериментов. Также данный класс используется для получения точек при построении графика.

```
1 public static double IntenseExperiment(double genIntense,
2   double procIntense, double procSigma, out string stat)
3   {
4       double sum = 0;
5       double load = 0;
6       double gen_i = 0;
7       double proc_i = 0;
8
9       for (int i = 0; i < repeats; i++)
10      {
11          ModellingController controller = new
12          ModellingController(true, 0, genIntense, true, 0, procSigma,
13          procIntense);
14          Report report = controller.StartModelling();
15          gen_i += report.GenIntense();
16          proc_i += report.ProcIntense();
17          load += report.Load();
18          sum += report.GetAvgTime();
19      }
20
21      stat = $"Параметры эксперимента: " +
22      $"интенсивность генератора {genIntense:F3},
23      интенсивность обработчика {procIntense:F3}, дисперсия
24      {procSigma:F3}\n" +
25      $"Ожидаемая загрузка: {genIntense /
26      procIntense:F3}, полученная загрузка: {load / repeats:F3}\n";
27      return sum / repeats;
28  }
```