Министерство образования и науки 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)

ОТЧЕТ
По лабораторной работе №2
По курсу «Планирование эксперимента»

Студент: Иванов Д. М.
Группа: ИУ7-81
Преподаватель: Куров А.В.

Москва, 2021.

Задание

1. Составить матрицу планирования, вывести на экран.
2. Провести опыты в точках плана.
3. Рассчитать коэффициенты линейной и частично-нелинейной моделей.
4. Предоставить возможность сравнить результаты в произвольных точках факторного пространства.

Законы распределения:

Генератор	Обслуживающий аппарат
Экспоненциальный	Нормальный

Эксперимент полнофакторный 2^3.

1. Аналитическая часть

Уравнение для линейной модели (для трех факторов):

$$y = b_0x_0 + b_1x_1 + b_2x_2 + b_3x_3$$

Уравнение для частично-нелинейной модели (для трех факторов):

$$y = b_0x_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_{12}x_1x_2 + b_{13}x_1x_3 + b_{23}x_2x_3 + b_{123}x_1x_2x_3$$

Матрица состоит из членов, входящих в данные уравнения. В таблице к этой матрице справа добавлются столбец с результатом, столбцы с расчетными результатами по данным уравнениям и столбцы, показывающие абсолютную разность между расчетными результатами и реальным.

Количество строк равно количеству экспериментов (2^3) плюс строки подэксперименты вне точек плана (не влияют на коэффициенты).

Коэффициенты рассчитываются по следующей формуле (в векторном виде):

$B = (X^TX)^{-1}X^TY$

$b_j = \frac{1}{N}\sum_{u=1}^{N} x_{ju}y_u$ (где значения x берутся из матрицы $(X^TX)^{-1}X^T$)

2. Технологическая часть

Программа выполнена на языке Python. Для получения законов распределения использовалась библиотека numpy. Для вывоа графика была использована библиотека matplotlib.

Листинг 1. main.py.

```
import sys
from PyQt5.QtWidgets import QApplication
from mainwindow import *


if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = MainWindow()
    sys.exit(app.exec_())
```

Листинг 2. Функции вычисления плана.

```
import numpy as np


def matrix_plan():
    res = [[1 for i in range(8)] for j in range(8)]
    step = 1
    for j in range(3):
        sign = -1
        for i in range(8):
            res[i][j + 1] = sign
            if (i + 1) % step == 0:
                sign *= -1
        step *= 2
    for i in range(8):
        res[i][4] = res[i][1] * res[i][2]
        res[i][5] = res[i][1] * res[i][3]
        res[i][6] = res[i][2] * res[i][3]
        res[i][7] = res[i][1] * res[i][2] * res[i][3]
    return res


def calc_xmat(plan):
    transposed = np.transpose(plan)
    mat = np.matmul(np.array(plan), transposed)
    mat = np.linalg.inv(mat)
    mat = np.matmul(mat, transposed)
    return mat.tolist()


def linear(b, x):
    res = 0
    linlen = int(np.log2(len(b))) + 1
    for i in range(linlen):
        res += b[i] * x[i]
    return res


def nonlinear(b, x):
    res = 0
```

```
    for i in range(len(b)):
        res += b[i] * x[i]
    return res


def expand_plan(plan, custom_plan, y, xmat):
    b = list()
    for i in range(len(xmat)):
        b_cur = 0
        for j in range(len(xmat[i])):
            b_cur += xmat[i][j] * y[j]
        b.append(b_cur)

    for i in custom_plan:
        if len(i) > 0:
            plan.append(i.copy())

    ylin = list()
    ynlin = list()
    for i in range(len(plan)):
        ylin.append(linear(b, plan[i]))
        ynlin.append(nonlinear(b, plan[i]))

    for i in range(len(plan)):
        plan[i].append(y[i])
        plan[i].append(ylin[i])
        plan[i].append(ynlin[i])
        plan[i].append(abs(y[i] - ylin[i]))
        plan[i].append(abs(y[i] - ynlin[i]))

    return plan, b


def scale_factor(x, realmin, realmax, xmin=-1, xmax=1):
    return realmin + (realmax - realmin) * (x - xmin) / (xmax - xmin)
```

Листинг 3. Код интерфейса

```
from PyQt5 import uic
from PyQt5.QtWidgets import QMainWindow, QTableWidgetItem, QHeaderView
from PyQt5.QtCore import Qt
from experiment_math import *
from model import *


class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        uic.loadUi('mainwindow.ui', self)
        self.show()
        self.almost_zero = 1e-10
        self.btn_do_plan.clicked.connect(self.do_plan)
        self.btn_set1.clicked.connect(self.set1)
        self.btn_set2.clicked.connect(self.set2)
        self.btn_set3.clicked.connect(self.set3)

self.table_plan.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

self.table_plan.verticalHeader().setSectionResizeMode(QHeaderView.Stretch)
        self.plan = matrix_plan()
        self.custom_plan = [list() for i in range(3)]
        for i in range(len(self.plan)):
```

```
            for j in range(len(self.plan[i])):
                self.table_plan.setItem(i + 1, j + 1,
QTableWidgetItem(str(round(self.plan[i][j], 3))))

    def get_factor(self, entry):
        try:
            res = float(entry.text())
        except ValueError:
            entry.setStyleSheet("background:#f88;")
            raise ValueError()

        entry.setStyleSheet("background:#fff;")

        if abs(res) < self.almost_zero:
            res = self.almost_zero

        return res

    @staticmethod
    def get_custom_factor(entry):
        try:
            res = float(entry.text())
            if not -1 <= res <= 1:
                raise ValueError()
            if res.is_integer():
                res = int(res)
        except ValueError:
            entry.setStyleSheet("background:#f88;")
            raise ValueError()

        entry.setStyleSheet("background:#fff;")
        return res

    def do_plan(self):
        self.plan = matrix_plan()
        self.xmat = calc_xmat(self.plan)
        total_apps = 10000
        try:
            gen_int_min = self.get_factor(self.entry_gen_int_min)
            gen_int_max = self.get_factor(self.entry_gen_int_max)
            proc_int_min = self.get_factor(self.entry_proc_int_min)
            proc_int_max = self.get_factor(self.entry_proc_int_max)
            proc_dev_min = self.get_factor(self.entry_proc_dev_min)
            proc_dev_max = self.get_factor(self.entry_proc_dev_max)
        except ValueError:
            pass
        else:
            y = list()
            # for each experiment
            for exp in self.plan:
                gen_int = scale_factor(exp[1], gen_int_min, gen_int_max)
                proc_int = scale_factor(exp[2], proc_int_min, proc_int_max)
                proc_dev = scale_factor(exp[3], proc_dev_min, proc_dev_max)

                gens = [Generator(exp_by_intensity, (gen_int,))]
                proc = Generator(norm_by_intensity, (proc_int, proc_dev))
                model = EventModel(gens, proc, total_apps)

                y.append(model.proceed() / total_apps)

            for exp in self.custom_plan:
                if len(exp) > 0:
                    gen_int = scale_factor(exp[1], gen_int_min, gen_int_max)
```

```
                    proc_int = scale_factor(exp[2], proc_int_min,
proc_int_max)
                    proc_dev = scale_factor(exp[3], proc_dev_min,
proc_dev_max)

                    gens = [Generator(exp_by_intensity, (gen_int,))]
                    proc = Generator(norm_by_intensity, (proc_int, proc_dev))
                    model = EventModel(gens, proc, total_apps)

                    y.append(model.proceed() / total_apps)

            old_size = len(self.plan)
            self.plan, b = expand_plan(self.plan, self.custom_plan, y,
self.xmat)

            for i in range(len(self.plan)):
                for j in range(old_size, len(self.plan[i])):
                    self.table_plan.setItem(i + 1, j + 1,
QTableWidgetItem(str(round(self.plan[i][j], 3))))

            self.set_equasions(b)

    def set_equasions(self, b, accuracy=3):
        yl = str(round(b[0], accuracy)) + " + " + str(round(b[1], accuracy))
+ "x1 + " + str(
            round(b[2], accuracy)) + "x2 + " + str(round(b[3], accuracy)) +
"x3"
        yl = yl.replace("+ -", "- ")
        ynl = yl + " + " + str(round(b[4], accuracy)) + "x1x2 + " +
str(round(b[5], accuracy)) + "x1x3 + " + str(
            round(b[6], accuracy)) + "x2x3 + " + str(round(b[7], accuracy)) +
"x1x2x3"
        ynl = ynl.replace("+ -", "- ")
        self.label_yl.setText(yl)
        self.label_ynl.setText(ynl)

    def set1(self):
        try:
            x1 = self.get_custom_factor(self.entry_x1_1)
            x2 = self.get_custom_factor(self.entry_x2_1)
            x3 = self.get_custom_factor(self.entry_x3_1)
        except ValueError:
            pass
        else:
            item = QTableWidgetItem("9")
            item.setTextAlignment(Qt.AlignCenter)
            self.table_plan.setItem(9, 0, item)
            self.custom_plan[0] = [1, x1, x2, x3, x1 * x2, x1 * x3, x2 * x3,
x1 * x2 * x3]
            for i in range(len(self.custom_plan[0])):
                self.table_plan.setItem(9, i + 1,
QTableWidgetItem(str(self.custom_plan[0][i])))
            for i in range(len(self.custom_plan[0]) + 1,
self.table_plan.columnCount()):
                self.table_plan.setItem(9, i, QTableWidgetItem(''))

    def set2(self):
        try:
            x1 = self.get_custom_factor(self.entry_x1_2)
            x2 = self.get_custom_factor(self.entry_x2_2)
            x3 = self.get_custom_factor(self.entry_x3_2)
        except ValueError:
            pass
        else:
```

```
            item = QTableWidgetItem("10")
            item.setTextAlignment(Qt.AlignCenter)
            self.table_plan.setItem(10, 0, item)
            self.custom_plan[1] = [1, x1, x2, x3, x1 * x2, x1 * x3, x2 * x3,
x1 * x2 * x3]
            for i in range(len(self.custom_plan[1])):
                self.table_plan.setItem(10, i + 1,
QTableWidgetItem(str(self.custom_plan[1][i])))

    def set3(self):
        try:
            x1 = self.get_custom_factor(self.entry_x1_3)
            x2 = self.get_custom_factor(self.entry_x2_3)
            x3 = self.get_custom_factor(self.entry_x3_3)
        except ValueError:
            pass
        else:
            item = QTableWidgetItem("11")
            item.setTextAlignment(Qt.AlignCenter)
            self.table_plan.setItem(11, 0, item)
            self.custom_plan[2] = [1, x1, x2, x3, x1 * x2, x1 * x3, x2 * x3,
x1 * x2 * x3]
            for i in range(len(self.custom_plan[2])):
                self.table_plan.setItem(11, i + 1,
QTableWidgetItem(str(self.custom_plan[2][i])))
```

Листинг 4. Модель

```
from numpy.random import exponential, normal


def exp_by_intensity(params):
    return exponential(1 / params[0])


def norm_by_intensity(params):
    res = -1
    while res < 0:
        res = normal(1 / params[0], 1 / params[1])
    return res


class Generator:
    def __init__(self, func, params):
        self.law = func
        self.params = params

    def generation_time(self):
        return self.law(self.params)


class EventModel:
    def __init__(self, generators, processor, total_apps=0):
        self.generators = generators
        self.processor = processor
        self.total_apps = total_apps

    def proceed(self):
        processed = 0
        self.queue = []
        self.events = []
        self.totally_waited = 0
```

```
        i = 0
        for generator in self.generators:
            self.events.append([generator.generation_time(), 'g', 0])
            i += 1
        self.free = True
        while processed < self.total_apps:
            event = self.events.pop(0)
            if event[1] == 'g':
                self._generate(event)
            elif event[1] == 'p':
                processed += 1
                self._process(event[0])

        return self.totally_waited

    def _add_event(self, event: list):
        i = 0
        while i < len(self.events) and self.events[i][0] < event[0]:
            i += 1
        self.events.insert(i, event)

    def _generate(self, event):
        self.queue.append(event[0])
        self._add_event([event[0] +
self.generators[event[2]].generation_time(), 'g', event[2]])
        if self.free:
            self._process(event[0])

    def _process(self, time):
        if len(self.queue) > 0:
            processing_time = self.processor.generation_time()
            self.totally_waited += processing_time + time - self.queue.pop(0)
            self._add_event([time + processing_time, 'p'])
            self.free = False
        else:
            self.free = True
```

3. Экспериментальная часть

На рисунке 1 показан результат работы программы

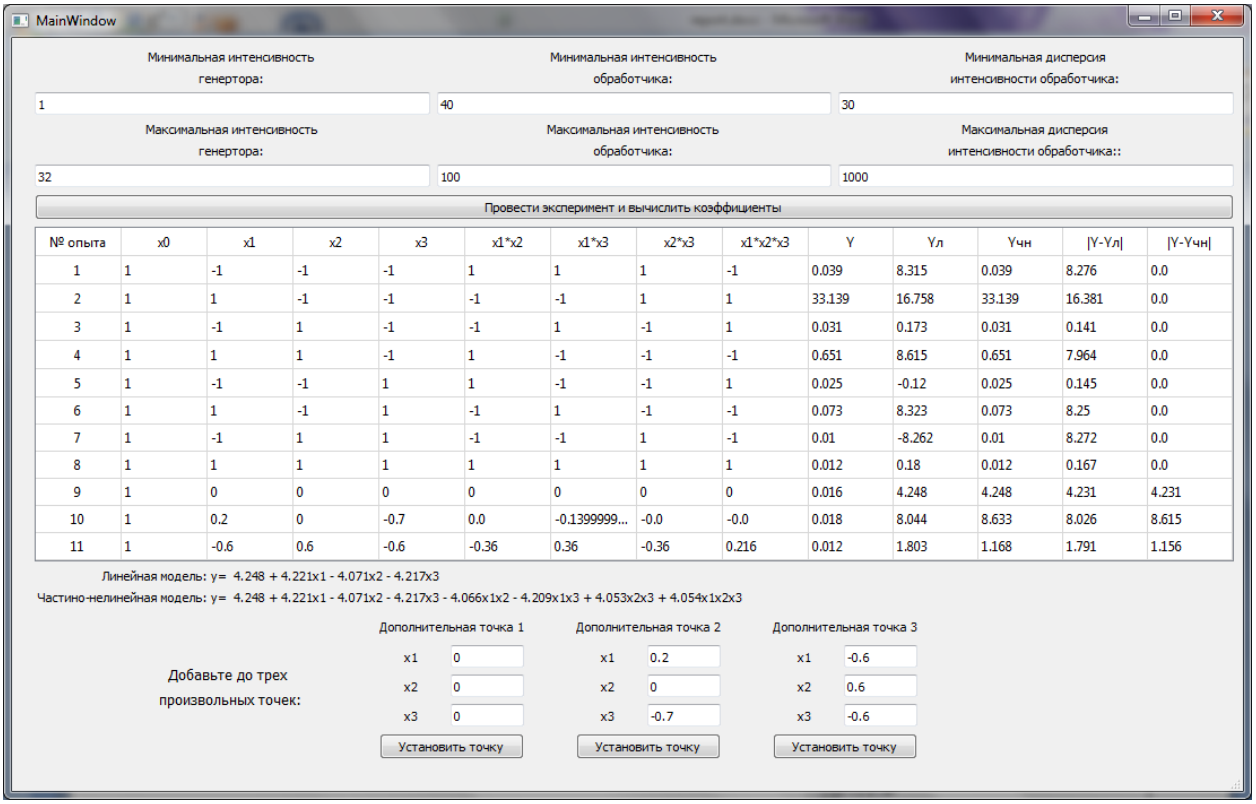

Рисунок 1. Результат эксперимента