|  |  |
| --- | --- |
|  | **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 23.

1. Аналитическая часть

Уравнение для линейной модели (для трех факторов):

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

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 (23) плюс строки подэксперименты вне точек плана (не влияют на коэффициенты).

Коэффициенты рассчитываются по следующей формуле (в векторном виде):

B = (XTX)-1XTY

(где значения x берутся из матрицы (XTX)-1XT)

1. 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'])

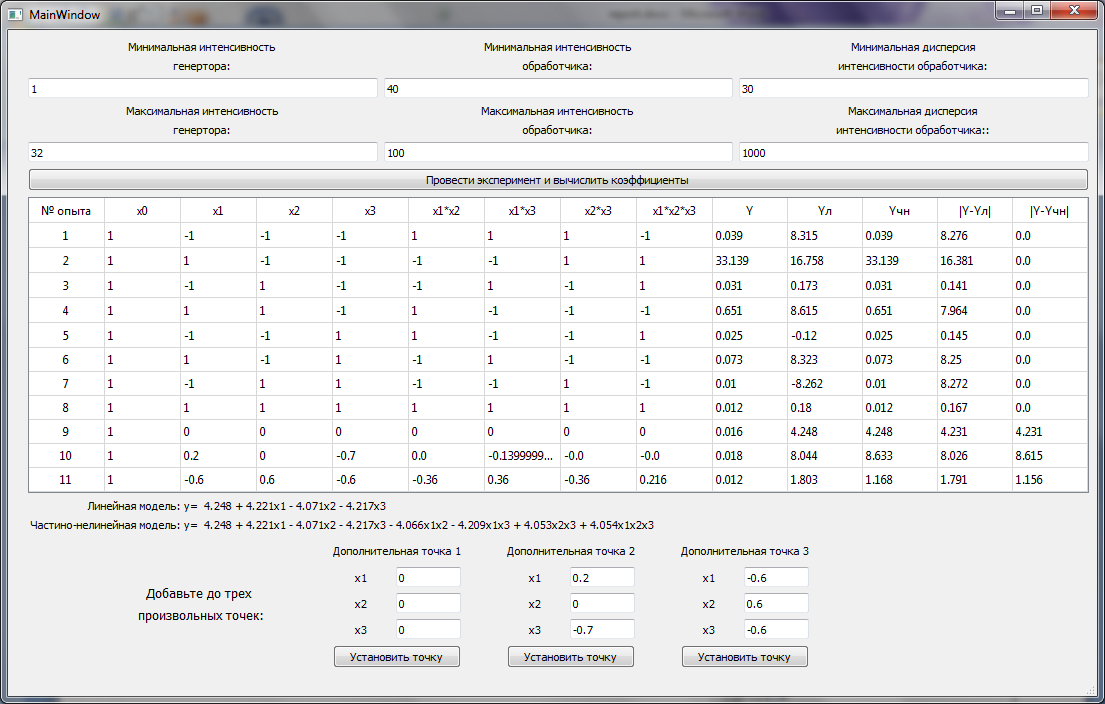
self.free = False

else:

self.free = True

1. Экспериментальная часть

На рисунке 1 показан результат работы программы

  
Рисунок 1. Результат эксперимента