Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки Кафедра
обчислювальної техніки

Методи планування експерименту

Лабораторна робота №5

«Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів (центральний ортогональний композиційний план)»

Виконав:
студент II курсу ФІОТ
групи ІВ-92
Салун Кирило
номер у списку групи – 20

Перевірив:
ас. Регіда П. Г.

Київ – 2021

Мета:

Провести трьохфакторний експеримент з урахуванням квадратичних членів ,використовуючи центральний ортогональний композиційний план. Знайти рівняння регресії, яке буде адекватним для опису об'єкту.

Завдання:

1. Взяти рівняння з урахуванням квадратичних членів.
2. Скласти матрицю планування для ОЦКП
3. Провести експеримент у всіх точках факторного простору (знайти значення функції відгуку Y). Значення функції відгуку знайти у відповідності з варіантом діапазону, зазначеного далі. Варіанти вибираються по номеру в списку в журналі викладача.

$$y_{i\max} = 200 + x_{cp\max}$$
$$y_{i\min} = 200 + x_{cp\min}$$

где $x_{cp\max} = \frac{x_{1\max} + x_{2\max} + x_{3\max}}{3}$, $x_{cp\min} = \frac{x_{1\min} + x_{2\min} + x_{3\min}}{3}$

4. Розрахувати коефіцієнти рівняння регресії і записати його.
5. Провести 3 статистичні перевірки.

Варіанти обираються по номеру в списку в журналі викладача.

Варіант завдання:

№варіанта	x_1		x_2		x_3	
	min	max	min	Max	min	max
220	-3	10	-1	2	-8	6

Лістинг програми:

```
import random
from functools import partial
 import numpy
import pandas
import sklearn.linear_model as lm
from scipy.stats import f, t from
pyDOE2 import ccdesign from
tabulate import tabulate

  class LaboratoryWorkN5:     def
__init__(self, x1, x2, x3):
        self.n, self.m = 15, 6
        self.x, self.y, self.x_normalized = None, None, None
self.y_average = None         self.b = None
        self.x_range = (x1, x2, x3)
        self.x_aver_max = numpy.average([x[1] for x in self.x_range])
self.x_aver_min = numpy.average([x[0] for x in self.x_range])
self.y_max = 200 + int(self.x_aver_max)         self.y_min = 200 +
int(self.x_aver_min)
     class Criteria:         def
__init__(self, x, y, n, m):
            self.x, self.y = x, y
self.n, self.m = n, m
            self.f1, self.f2 = self.m - 1, self.n
self.q = 0.05
            self.q1 = self.q / self.f1
         def s_kv(self,
y_average):
            result = []
for i in range(self.n):
                s = sum([(y_average[i] - self.y[i][j]) ** 2 for j in
range(self.m)]) / self.m
                result.append(round(s, 3))
return result
         def cochrane_criterion(self,
y_average):
            s_kv = self.s_kv(y_average)
gp = max(s_kv) / sum(s_kv)
            print('Перевірка за критерієм Кохрена:')
return gp
         def
cochrane(self):
            fisher_value = f.ppf(q=1 - self.q1, dfn=self.f2, dfd=(self.f1
- 1) * self.f2)
            return fisher_value / (fisher_value + self.f1 - 1)
         def bs(self, x,
y_average):
            result = [sum(y_average) / self.n]
for i in range(len(x[0])):
                b = sum(j[0] * j[1] for j in zip(x[:, i], y_average)) /
self.n
                result.append(b)
return result

        def student_criterion(self, x, y_average):
s_kv = self.s_kv(y_average)             s_kv_aver =
sum(s_kv) / self.n             s_bs = (s_kv_aver /
```

```
self.n / self.m) ** 0.5                bs = self.bs(x,
y_average)
            return [round(abs(B) / s_bs, 3) for B in bs]
        def fisher_criterion(self, y_average, y_new,
d):
            s_ad = self.m / (self.n - d) * sum([(y_new[i] - y_average[i])
** 2 for i in range(len(self.y))])
s_kv = self.s_kv(y_average)
s_kv_aver = sum(s_kv) / self.n
return s_ad / s_kv_aver

    @staticmethod     def
add_sq_nums(x):          for i
in range(len(x)):
            x[i][4] = x[i][1] * x[i][2]
x[i][5] = x[i][1] * x[i][3]              x[i][6] =
x[i][2] * x[i][3]              x[i][7] = x[i][1] *
x[i][3] * x[i][2]              x[i][8] = x[i][1]
** 2              x[i][9] = x[i][2] ** 2
x[i][10] = x[i][3] ** 2          return x
    def
get_y_average(self):
        self.y_average = [round(sum(i) / len(i), 3) for i in self.y]
    @staticmethod     def
regression_equation(x, b):
        return sum([x[i] * b[i] for i in range(len(x))])
    def
get_b_coefficient(self):
        skm = lm.LinearRegression(fit_intercept=False)
skm.fit(self.x, self.y_average)          self.b =
skm.coef_
        print('Коефіцієнти рівняння регресії:')
self.b = [round(i, 3) for i in self.b]
        print('\ty = {} +{}*x1 +{}*x2 +{}*x3 + {}*x1*x2 + {}*x1*x3 +
{}*x2*x3 + b{}*x1*x2*x3 + {}x1^2 + {}x2^2 + {}x3^2'
              .format(*self.b))
        print(f'Результат рівняння зі знайденими
коефіцієнтами:\n\t{numpy.dot(self.x, self.b)}')
    def
check(self):
        criteria = self.Criteria(self.x, self.y, self.n, self.m)

        print('Перевірка рівняння:')
f1 = self.m - 1          f2 = self.n
f3 = f1 * f2          q = 0.05

        student = partial(t.ppf, q=1 - q)
t_student = student(df=f3)          g_kr =
criteria.cochrane()
        y_average = [round(sum(i) / len(i), 3) for i in self.y]
print(f'\tСереднє значення y: {y_average}')          dispersion
= criteria.s_kv(y_average) print(f'\tДисперсія y:
{dispersion}')          gp =
criteria.cochrane_criterion(y_average)          print(f'\tGp =
{gp}')          if gp < g_kr:
```

```
            print(f'\tЗ імовірністю {1 - q} дисперсії однорідні.')
else:               print('\tНеобхідно збільшити кількість
дослідів!')                 self.m += 1
            new_exp = LaboratoryWorkN5(*self.x_range)
new_exp.run(self.n, self.m)

        ts = criteria.student_criterion(self.x_normalized[:, 1:],
y_average)
        print(f'Перевірка за критерієм Стьюдента:\n\t{ts}')
result = [element for element in ts if element > t_student]
final_k = [self.b[i] for i in range(len(ts)) if ts[i] in result]
print(f'\tКоефіцієнти {[round(i, 3) for i in self.b if i not in
final_k]} '
              f'статистично незначущі, тому ми виключаємо їх з
рівняння.')

        y_new = []          for
j in range(self.n):
y_new.append(round(
                self.regression_equation([self.x[j][i] for i in
range(len(ts)) if ts[i] in result], final_k), 3))

        print(f'Значення Y з коефіцієнтами {final_k}:')
print(f'\t{y_new}')

        d = len(result)
if d >= self.n:
            print('F4 <= 0')

        f4 = self.n - d

        f_p = criteria.fisher_criterion(y_average, y_new, d)

        fisher = partial(f.ppf, q=0.95)
f_t = fisher(dfn=f4, dfd=f3)
        print('Перевірка адекватності за критерієм Фішера:')
print(f'\tFp = {f_p}')          print(f'\tFt = {f_t}')          if
f_p < f_t:              print('\tМатематична модель адекватна
експериментальним даним.')          else:
            print('\tМатематична модель не адекватна експериментальним
даним.')       def fill_y(self):
        self.y = numpy.zeros(shape=(self.n, self.m))
for i in range(self.n):              for j in
range(self.m):
                self.y[i][j] = random.randint(self.y_min, self.y_max)
    def
form_matrix(self):
        self.fill_y()
if self.n > 14:
            no = self.n - 14
        else:
no = 1

        self.x_normalized = ccdesign(3, center=(0, no))
        self.x_normalized = numpy.insert(self.x_normalized, 0, 1, axis=1)
         for i in range(4,
11):
```

```
            self.x_normalized = numpy.insert(self.x_normalized, i, 0,
axis=1)
        lc = 1.215
         for i in range(len(self.x_normalized)):
for j in range(len(self.x_normalized[i])):
                if self.x_normalized[i][j] < -1 or
self.x_normalized[i][j] > 1:                    if
self.x_normalized[i][j] < 0:
self.x_normalized[i][j] = -lc
else:
                        self.x_normalized[i][j] = lc

        self.x_normalized = self.add_sq_nums(self.x_normalized)

        self.x = numpy.ones(shape=(len(self.x_normalized),
len(self.x_normalized[0])), dtype=numpy.int64)         for
i in range(8):             for j in range(1, 4):
if self.x_normalized[i][j] == -1:
                    self.x[i][j] = self.x_range[j - 1][0]
else:
                    self.x[i][j] = self.x_range[j - 1][1]
         for i in range(8,
len(self.x)):             for j in
range(1, 3):
                self.x[i][j] = (self.x_range[j - 1][0] + self.x_range[j -
1][1]) / 2

        dx = [self.x_range[i][1] - (self.x_range[i][0] +
self.x_range[i][1]) / 2 for i in range(3)]

        self.x[8][1] = lc * dx[0] + self.x[9][1]
self.x[9][1] = -lc * dx[0] + self.x[9][1]
self.x[10][2] = lc * dx[1] + self.x[9][2]
self.x[11][2] = -lc * dx[1] + self.x[9][2]
self.x[12][3] = lc * dx[2] + self.x[9][3]
self.x[13][3] = -lc * dx[2] + self.x[9][3]         self.x
= self.add_sq_nums(self.x)

        show_arr = pandas.DataFrame(self.x)
print('X:\n', tabulate(show_arr, headers='keys',
tablefmt='psql'))

        show_arr = pandas.DataFrame(self.x_normalized)
print('Нормовані X:\n', tabulate(show_arr.round(0), headers='keys',
tablefmt='psql'))

        show_arr = pandas.DataFrame(self.y)
print('Y:\n', tabulate(show_arr, headers='keys',
tablefmt='psql'))

    def run(self, n=None, m=None):
if n is not None and m is not None:
            self.n = n
self.m = m
```

```
        self.form_matrix()
self.get_y_average()
self.get_b_coefficient()          self.check()
  if __name__ ==
'__main__':
    worker = LaboratoryWorkN5((-6, 10), (-8, 3), (-10, 9))
worker.run(15, 3)
```

Результати виконання роботи:

Гереруємо матрицю планування для n = 15, m = 3

X:

```
 [[   1   -3   -1   -8    3   24    8  -24    9    1   64]
 [   1   10   -1   -8  -10  -80    8   80  100    1   64]
 [   1   -3    2   -8   -6   24  -16   48    9    4   64]
 [   1   10    2   -8   20  -80  -16 -160  100    4   64]
 [   1   -3   -1    6    3  -18   -6   18    9    1   36]
 [   1   10   -1    6  -10   60   -6  -60  100    1   36]
 [   1   -3    2    6   -6  -18   12  -36    9    4   36]
 [   1   10    2    6   20   60   12  120  100    4   36]
 [   1   10    0    1    0   10    0    0  100    0    1]
 [   1   -4    0    1    0   -4    0    0   16    0    1]
 [   1    3    1    1    3    3    1    3    9    1    1]
 [   1    3   -1    1   -3    3   -1   -3    9    1    1]
 [   1    3    0    9    0   27    0    0    9    0   81]
 [   1    3    0   -7    0  -21    0    0    9    0   49]
 [   1    3    0    1    0    3    0    0    9    0    1]]
```

X нормоване:

```
[1.0, -1.0, -1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, -1.0, -1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, -1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, -1.0, 1.0, 1.0, 1.0]
[1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, -1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0]
[1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, -1.22, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 1.48, 0.0, 0.0]
[1.0, 1.22, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.48, 0.0, 0.0]
[1.0, 0.0, -1.22, 0.0, -0.0, 0.0, -0.0, -0.0, 0.0, 1.48, 0.0]
[1.0, 0.0, 1.22, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.48, 0.0]
[1.0, 0.0, 0.0, -1.22, 0.0, -0.0, -0.0, -0.0, 0.0, 0.0, 1.48]
[1.0, 0.0, 0.0, 1.22, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.48]
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

```
Y:
 [[206. 196. 198.]
 [204. 203. 205.]
 [198. 203. 204.]
 [198. 202. 201.]
 [206. 206. 196.]
 [199. 203. 206.]
 [196. 200. 197.]
 [200. 200. 198.]
 [197. 205. 204.]
 [198. 201. 201.]
 [198. 204. 204.]
 [205. 205. 199.]
 [196. 196. 200.]
 [199. 200. 202.]
 [200. 198. 205.]]

Коефіцієнти рівняння регресії:
[201.179, 0.128, -1.41, -0.081, -0.038, -0.009, -0.113, 0.013, -0.002, 0.758, -0.036]

Результат рівняння зі знайденими коефіцієнтами:
 [199.743 204.007 201.777 200.503 202.123 202.383 197.773 199.593 202.052
 200.554 200.561 203.757 197.657 200.537 201.401]
```

```
   Перевірка рівняння:

Середнє значення y: [200.0, 204.0, 201.667, 200.333, 202.667, 202.667, 197.667, 199.333, 202.0, 200.0, 202.0, 203.0, 197.333, 200.333, 201.0]
Дисперсія y: [18.667, 0.667, 6.889, 2.889, 22.222, 8.222, 2.889, 0.889, 12.667, 2.0, 8.0, 8.0, 3.556, 1.556, 8.667]

Перевірка за критерієм Кохрена
Gp = 0.20617925403599927
З ймовірністю 0.95 дисперсії однорідні.

Критерій Стьюдента:
 [502.846, 0.317, 1.521, 0.004, 0.612, 0.167, 1.057, 1.168, 367.337, 368.076, 366.27]

Коефіцієнти [0.128, -1.41, -0.081, -0.038, -0.009, -0.113, 0.013] статистично незначущі, тому ми виключаємо їх з рівняння.

Значення "y" з коефіцієнтами [201.179, -0.002, 0.758, -0.036]
[201.899, 201.899, 201.899, 201.899, 201.899, 201.899, 201.899, 201.899, 201.17604755, 201.17604755, 202.29797855, 202.29797855, 201.1258559, 201.1258559, 201.179]

Перевірка адекватності за критерієм Фішера
Fp = 2.0454711766272493
F_t = 2.125558760875511
Математична модель адекватна експериментальним даним
```

Висновок:

У ході виконання лабораторної роботи проведено повний трьохфакторний експеримент. В ході дослідження було розроблено відповідну програму мовою програмування Python, яка моделює проведення трьохфакторного експерименту, використовуючи центральний ортогональний композиційний план. Знайдено рівняння регресії, адекватне для опису об’єкту, проведено 3

статистичні перевірки (критерії Кохрена, Стьюдента та Фішера). Результати роботи, наведені у протоколі, підтверджують правильність виконання – кінцеву мету роботи було досягнуто.