

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

ЛАБОРАТОРНА РОБОТА №6

з дисципліни « Методи оптимізації та планування » на тему
«Проведення трьохфакторного експерименту при використанні рівняння
регресії з квадратичними членами»

Виконала:
студентка II курсу ФІОТ
групи ІВ – 92
Гайдукевич Марія
Номер залікової книжки: 9206

Перевірів:
ас. Регіда П.Г.

Мета роботи: провести трьохфакторний експеримент і отримати адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план.

Завдання на лабораторну роботу:

1. Ознайомитися з теоретичними відомостями.
2. Вибрати з таблиці варіантів і записати в протокол інтервали значень x_1 , x_2 , x_3 . Обчислити і записати значення, відповідні кодованим значенням факторів +1; -1; +; -; 0 для 1, 2, 3.
3. Значення функції відгуку знайти за допомогою підстановки в формулу: $y_i = f(x_1, x_2, x_3) + \text{random}(10) - 5$, де $f(x_1, x_2, x_3)$ вибирається по номеру в списку в журналі викладача.
4. Провести експерименти і аналізуючи значення статистичних перевірок, отримати адекватну модель рівняння регресії. При розрахунках використовувати натуральні значення факторів.
5. Зробити висновки по виконаній роботі.

Варіант: 204

Роздруківка тексту програми:

```
from math import fabs, sqrt

m = 3
p = 0.95
N = 15
x1_min = 15
x1_max = 45
x2_min = 15
x2_max = 50
x3_min = 15
x3_max = 30
x01 = (x1_max + x1_min) / 2
x02 = (x2_max + x2_min) / 2
x03 = (x3_max + x3_min) / 2
delta_x1 = x1_max - x01
delta_x2 = x2_max - x02
delta_x3 = x3_max - x03

def get_cohren_value(size_of_selections, qty_of_selections, significance):
    from _pydecimal import Decimal
    from scipy.stats import f
    size_of_selections += 1
    partResult1 = significance / (size_of_selections - 1)
    params = [partResult1, qty_of_selections, (size_of_selections - 1 - 1) *
qty_of_selections]
    fisher = f.isf(*params)
    result = fisher / (fisher + (size_of_selections - 1 - 1))
    return Decimal(result).quantize(Decimal('.0001')).__float__()

def get_student_value(f3, significance):
    from _pydecimal import Decimal
    from scipy.stats import t
    return Decimal(abs(t.ppf(significance / 2,
```

```
f3))).quantize(Decimal('.0001')).__float__()
```

```
def get_fisher_value(f3, f4, significance):  
    from pydecimal import Decimal  
    from scipy.stats import f  
    return Decimal(abs(f.isf(significance, f4,  
f3))).quantize(Decimal('.0001')).__float__()
```

```
def generate_matrix():  
    def f(X1, X2, X3):  
        from random import randrange  
        y = 3.5 + 6.6 * X1 + 3.9 * X2 + 1.8 * X3 + 5.3 * X1 * X1 + 0.5 * X2 * X2 +  
4.3 * X3 * X3 + \  
        6.0 * X1 * X2 + 0.8 * X1 * X3 + 9.4 * X2 * X3 + 3.0 * X1 * X2 * X3 +  
randrange(0, 10) - 5  
        return y  
  
    matrix_with_y = [[f(matrix_x[j][0], matrix_x[j][1], matrix_x[j][2]) for _ in  
range(m)] for j in range(N)]  
    return matrix_with_y
```

```
def x(l1, l2, l3):  
    x_1 = l1 * delta_x1 + x01  
    x_2 = l2 * delta_x2 + x02  
    x_3 = l3 * delta_x3 + x03  
    return [x_1, x_2, x_3]
```

```
def find_average(lst, orientation):  
    average = []  
    if orientation == 1:  
        for rows in range(len(lst)):  
            average.append(sum(lst[rows]) / len(lst[rows]))  
    else:  
        for column in range(len(lst[0])):  
            number_lst = []  
            for rows in range(len(lst)):  
                number_lst.append(lst[rows][column])  
            average.append(sum(number_lst) / len(number_lst))  
    return average
```

```
def a(first, second):  
    need_a = 0  
    for j in range(N):  
        need_a += matrix_x[j][first - 1] * matrix_x[j][second - 1] / N  
    return need_a
```

```
def find_known(number):  
    need_a = 0  
    for j in range(N):  
        need_a += average_y[j] * matrix_x[j][number - 1] / 15  
    return need_a
```

```
def solve(lst_1, lst_2):  
    from numpy.linalg import solve  
    solver = solve(lst_1, lst_2)
```

```

    return solver

def check_result(b_lst, k):
    y_i = b_lst[0] + b_lst[1] * matrix[k][0] + b_lst[2] * matrix[k][1] + b_lst[3] *
matrix[k][2] + \
        b_lst[4] * matrix[k][3] + b_lst[5] * matrix[k][4] + b_lst[6] * matrix[k][5]
+ b_lst[7] * matrix[k][6] + \
        b_lst[8] * matrix[k][7] + b_lst[9] * matrix[k][8] + b_lst[10] *
matrix[k][9]
    return y_i

def student_test(b_lst, number_x=10):
    dispersion_b = sqrt(dispersion_b2)
    for column in range(number_x + 1):
        t_practice = 0
        t_theoretical = get_student_value(f3, q)
        for row in range(N):
            if column == 0:
                t_practice += average_y[row] / N
            else:
                t_practice += average_y[row] * matrix_pfe[row][column - 1]
            if fabs(t_practice / dispersion_b) < t_theoretical:
                b_lst[column] = 0
    return b_lst

def fisher_test():
    dispersion_ad = 0
    f4 = N - d
    for row in range(len(average_y)):
        dispersion_ad += (m * (average_y[row] - check_result(student_lst, row))) / (N
- d)
    F_practice = dispersion_ad / dispersion_b2
    F_theoretical = get_fisher_value(f3, f4, q)
    return F_practice < F_theoretical

matrix_pfe = [
    [-1, -1, -1, +1, +1, +1, -1, +1, +1, +1],
    [-1, -1, +1, +1, -1, -1, +1, +1, +1, +1],
    [-1, +1, -1, -1, +1, -1, +1, +1, +1, +1],
    [-1, +1, +1, -1, -1, +1, -1, +1, +1, +1],
    [+1, -1, -1, -1, -1, +1, +1, +1, +1, +1],
    [+1, -1, +1, -1, +1, -1, -1, +1, +1, +1],
    [+1, +1, -1, +1, -1, -1, -1, +1, +1, +1],
    [+1, +1, +1, +1, +1, +1, +1, +1, +1, +1],
    [-1.73, 0, 0, 0, 0, 0, 0, 0, 2.9929, 0, 0],
    [+1.73, 0, 0, 0, 0, 0, 0, 0, 2.9929, 0, 0],
    [0, -1.73, 0, 0, 0, 0, 0, 0, 0, 2.9929, 0],
    [0, +1.73, 0, 0, 0, 0, 0, 0, 0, 2.9929, 0],
    [0, 0, -1.73, 0, 0, 0, 0, 0, 0, 0, 2.9929],
    [0, 0, +1.73, 0, 0, 0, 0, 0, 0, 0, 2.9929],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]

matrix_x = [[] for _ in range(N)]
for i in range(len(matrix_x)):
    if i < 8:
        x_1 = x1_min if matrix_pfe[i][0] == -1 else x1_max
        x_2 = x2_min if matrix_pfe[i][1] == -1 else x2_max

```

```

        x_3 = x3_min if matrix_pfe[i][2] == -1 else x3_max
    else:
        x_lst = x(matrix_pfe[i][0], matrix_pfe[i][1], matrix_pfe[i][2])
        x_1, x_2, x_3 = x_lst
        matrix_x[i] = [x_1, x_2, x_3, x_1 * x_2, x_1 * x_3, x_2 * x_3, x_1 * x_2 * x_3,
x_1 ** 2, x_2 ** 2, x_3 ** 2]

adekvat = False
odnorid = False
while not adekvat:
    matrix_y = generate_matrix()
    average_x = find_average(matrix_x, 0)
    average_y = find_average(matrix_y, 1)
    matrix = [(matrix_x[i] + matrix_y[i]) for i in range(N)]
    mx_i = average_x
    my = sum(average_y) / 15

    unknown = [
        [1, mx_i[0], mx_i[1], mx_i[2], mx_i[3], mx_i[4], mx_i[5], mx_i[6], mx_i[7],
mx_i[8], mx_i[9]],
        [mx_i[0], a(1, 1), a(1, 2), a(1, 3), a(1, 4), a(1, 5), a(1, 6), a(1, 7), a(1,
8), a(1, 9), a(1, 10)],
        [mx_i[1], a(2, 1), a(2, 2), a(2, 3), a(2, 4), a(2, 5), a(2, 6), a(2, 7), a(2,
8), a(2, 9), a(2, 10)],
        [mx_i[2], a(3, 1), a(3, 2), a(3, 3), a(3, 4), a(3, 5), a(3, 6), a(3, 7), a(3,
8), a(3, 9), a(3, 10)],
        [mx_i[3], a(4, 1), a(4, 2), a(4, 3), a(4, 4), a(4, 5), a(4, 6), a(4, 7), a(4,
8), a(4, 9), a(4, 10)],
        [mx_i[4], a(5, 1), a(5, 2), a(5, 3), a(5, 4), a(5, 5), a(5, 6), a(5, 7), a(5,
8), a(5, 9), a(5, 10)],
        [mx_i[5], a(6, 1), a(6, 2), a(6, 3), a(6, 4), a(6, 5), a(6, 6), a(6, 7), a(6,
8), a(6, 9), a(6, 10)],
        [mx_i[6], a(7, 1), a(7, 2), a(7, 3), a(7, 4), a(7, 5), a(7, 6), a(7, 7), a(7,
8), a(7, 9), a(7, 10)],
        [mx_i[7], a(8, 1), a(8, 2), a(8, 3), a(8, 4), a(8, 5), a(8, 6), a(8, 7), a(8,
8), a(8, 9), a(8, 10)],
        [mx_i[8], a(9, 1), a(9, 2), a(9, 3), a(9, 4), a(9, 5), a(9, 6), a(9, 7), a(9,
8), a(9, 9), a(9, 10)],
        [mx_i[9], a(10, 1), a(10, 2), a(10, 3), a(10, 4), a(10, 5), a(10, 6), a(10,
7), a(10, 8), a(10, 9), a(10, 10)]
    ]
    known = [my, find_known(1), find_known(2), find_known(3), find_known(4),
find_known(5), find_known(6),
        find_known(7),
        find_known(8), find_known(9), find_known(10)]

    beta = solve(unknown, known)
    print("Отримане рівняння регресії")
    print("{:.3f} + {:.3f} * X1 + {:.3f} * X2 + {:.3f} * X3 + {:.3f} * X1X2 + {:.3f}
* X1X3 + {:.3f} * X2X3"
        "+ {:.3f} * X1X2X3 + {:.3f} * X11^2 + {:.3f} * X22^2 + {:.3f} * X33^2 =
ŷ\n\tПевірка"
        .format(beta[0], beta[1], beta[2], beta[3], beta[4], beta[5], beta[6],
beta[7], beta[8], beta[9], beta[10]))
    for i in range(N):
        print("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1), check_result(beta, i),
average_y[i]))

    dispersion_y = [0.0 for x in range(N)]
    while not odnorid:
        print("Матриця планування експерименту:")
        print("
            X1            X2            X3            X1X2            X1X3

```

```

X2X3      X1X2X3      X1X1"
"          X2X2      X3X3      Yi ->")
for row in range(N):
    print(end=' ')
    for column in range(len(matrix[0])):
        print("{:^12.3f}".format(matrix[row][column]), end=' ')
    print("")

    for i in range(N):
        dispersion_i = 0
        for j in range(m):
            dispersion_i += (matrix_y[i][j] - average_y[i]) ** 2
        dispersion_y.append(dispersion_i / (m - 1))
f1 = m - 1
f2 = N
f3 = f1 * f2
q = 1 - p
Gp = max(dispersion_y) / sum(dispersion_y)
print("Критерій Кохрена:")
Gt = get_cohren_value(f2, f1, q)
if Gt > Gp:
    print("Дисперсія однорідна при рівні значимості {:.2f}.".format(q))
    odnorid = True
else:
    print("Дисперсія не однорідна при рівні значимості {:.2f}! Збільшуємо
m.".format(q))
    m += 1

dispersion_b2 = sum(dispersion_y) / (N * N * m)
student_lst = list(student_test(beta))
print("Отримане рівняння регресії з урахуванням критерія Стьюдента")
print("{:.3f} + {:.3f} * X1 + {:.3f} * X2 + {:.3f} * X3 + {:.3f} * X1X2 + {:.3f}
* X1X3 + {:.3f} * X2X3"
      + {:.3f} * X1X2X3 + {:.3f} * X11^2 + {:.3f} * X22^2 + {:.3f} * X33^2 =
ŷ\n\tПеревірка"
      .format(student_lst[0], student_lst[1], student_lst[2], student_lst[3],
student_lst[4], student_lst[5],
student_lst[6], student_lst[7], student_lst[8], student_lst[9],
student_lst[10]))
for i in range(N):
    print("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1), check_result(student_lst, i),
average_y[i]))

print("Критерій Фішера")
d = 11 - student_lst.count(0)
if fisher_test():
    print("Рівняння регресії адекватне оригіналу")
    adekvat = True
else:
    print("Рівняння регресії неадекватне оригіналу\n\tПроводимо експеримент
повторно")

```

Результати роботи програми:

```
Отримане рівняння регресії
-13.369 + 7.000 * X1 + 4.177 * X2 + 2.948 * X3 + 5.993 * X1X2 + 0.794 * X1X3 + 9.395 * X2X3+ 3.000 * X1X2X3 + 5.295 * X11^2 + 0.498 * X22^2 + 4.277 * X33^2 = ŷ

Перевірка
ŷ1 = 16230.977 ≈ 16230.833
ŷ2 = 31580.042 ≈ 31580.333
ŷ3 = 49216.160 ≈ 49216.500
ŷ4 = 93124.225 ≈ 93125.000
ŷ5 = 49278.293 ≈ 49277.833
ŷ6 = 85236.025 ≈ 85236.000
ŷ7 = 135809.810 ≈ 135809.833
ŷ8 = 247579.875 ≈ 247580.333
ŷ9 = 19611.744 ≈ 19611.167
ŷ10 = 161364.743 ≈ 161364.899
ŷ11 = 13117.725 ≈ 13118.074
ŷ12 = 161640.272 ≈ 161639.502
ŷ13 = 42881.794 ≈ 42882.086
ŷ14 = 132402.776 ≈ 132402.063
ŷ15 = 86922.330 ≈ 86922.333

Матриця планування експерименту:
X1      X2      X3      X1X2      X1X3      X2X3      X1X2X3      X1X1      X2X2      X3X3      Yi ->
15.000  15.000  15.000  225.000  225.000  225.000  3375.000  225.000  225.000  225.000  16232.500  16234.500  16225.500
15.000  15.000  30.000  225.000  450.000  450.000  6750.000  225.000  225.000  900.000  31582.000  31584.000  31575.000
15.000  50.000  15.000  750.000  225.000  750.000  11250.000  225.000  2500.000  225.000  49215.500  49217.500  49216.500
15.000  50.000  30.000  750.000  450.000  1500.000  22500.000  225.000  2500.000  900.000  93127.000  93124.000  93124.000
45.000  15.000  15.000  675.000  675.000  225.000  10125.000  2025.000  225.000  225.000  49276.500  49282.500  49274.500
45.000  15.000  30.000  675.000  1350.000  450.000  20250.000  2025.000  225.000  900.000  85236.000  85234.000  85238.000
45.000  50.000  15.000  2250.000  675.000  750.000  33750.000  2025.000  2500.000  225.000  135811.500  135810.500  135807.500
45.000  50.000  30.000  2250.000  1350.000  1500.000  67500.000  2025.000  2500.000  900.000  247580.000  247581.000  247580.000
4.050   32.500  22.500  131.625  91.125  731.250  2961.563  16.403  1056.250  506.250  19606.501  19613.501  19613.501
55.950  32.500  22.500  1818.375  1258.875  731.250  40913.438  3130.403  1056.250  506.250  161364.566  161365.566  161364.566
30.000  2.225  22.500  66.750  675.000  50.063  1501.875  900.000  4.951  506.250  13117.740  13118.740  13117.740
30.000  62.775  22.500  1883.250  675.000  1412.438  42373.125  900.000  3940.701  506.250  161636.835  161638.835  161642.835
30.000  32.500  9.525  975.000  285.750  309.562  9286.875  900.000  1056.250  90.726  42881.753  42884.753  42879.753
30.000  32.500  35.475  975.000  1064.250  1152.938  34588.125  900.000  1056.250  1258.476  132405.063  132400.063  132401.063
30.000  32.500  22.500  975.000  675.000  731.250  21937.500  900.000  1056.250  506.250  86922.000  86923.000  86922.000

Критерій Кохрена:
Дисперсія однорідна при рівні значимості 0.05.
Отримане рівняння регресії з урахуванням критерія Стьюдента
-13.369 + 7.000 * X1 + 4.177 * X2 + 2.948 * X3 + 5.993 * X1X2 + 0.794 * X1X3 + 9.395 * X2X3+ 3.000 * X1X2X3 + 5.295 * X11^2 + 0.498 * X22^2 + 4.277 * X33^2 = ŷ

Перевірка
ŷ1 = 16230.977 ≈ 16230.833
ŷ2 = 31580.042 ≈ 31580.333
ŷ3 = 49216.160 ≈ 49216.500
ŷ4 = 93124.225 ≈ 93125.000
ŷ5 = 49278.293 ≈ 49277.833
ŷ6 = 85236.025 ≈ 85236.000
ŷ7 = 135809.810 ≈ 135809.833
ŷ8 = 247579.875 ≈ 247580.333
ŷ9 = 19611.744 ≈ 19611.167
ŷ10 = 161364.743 ≈ 161364.899
ŷ11 = 13117.725 ≈ 13118.074
ŷ12 = 161640.272 ≈ 161639.502
ŷ13 = 42881.794 ≈ 42882.086
ŷ14 = 132402.776 ≈ 132402.063
ŷ15 = 86922.330 ≈ 86922.333
Критерій Фішера
Рівняння регресії адекватне оригіналу

Process finished with exit code 0
```



Висновки:

Під час виконання лабораторної роботи було змодельовано трьохфакторний експеримент при використанні лінійного рівняння регресії, рівняння регресії з ефектом взаємодії та рівняння регресії з квадратичними членами, складено матрицю планування експерименту, було визначено коефіцієнти рівнянь регресії (натуралізовані та нормовані), для форми з квадратичними членами - натуралізовані, виконано перевірку правильності розрахунку коефіцієнтів рівнянь регресії. Також було проведено 3 статистичні перевірки(використання критеріїв Кохрена, Стьюдента та Фішера) для кожної форми рівняння регресії. Мета лабораторної роботи досягнена.