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

Лабораторна робота №6

З дисципліни «Методи оптимізації та планування»

Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами

ВИКОНАВ:
Студент II курсу ФІОТ
Групи ІО-93
Корякін Є. Ю. - 9317

ПЕРЕВІРИВ:
Регіда П.Г.

Київ 2021 р.

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

Варіант завдання:

315	10	50	25	65	50	65	7,9+2,1*x1+5,3*x2+3,0*x3+8,1*x1*x1+1,0*x2*x2+8,4*x3*x3+7,2*x1*x2+0,8*x1*x3+2,3*x2*x3+6,4*x1*x2*x3

Лістинг програми:

```
from math import fabs, sqrt
import time
m = 2
p = 0.95
N = 15
x1_min = 10
x1_max = 50
x2_min = 25
x2_max = 65
x3_min = 50
x3_max = 65
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

average_y = None
matrix = None
dispersion_b2 = None
student_lst = None
d = None
q = None
f3 = None


class Perevirku:
    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 =
7.9+2.1*X1+5.3*X2+3.0*X3+8.1*X1*X1+1.0*X2*X2+8.4*X3*X3+7.2*X1*X2+0.8*X1*X3+2.3*X2*X3+
6.4*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 i 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 = Perevirku.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 = Perevirku.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, 2.9929, 0, 0],
    [+1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0, 0],
    [0, -1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0],
    [0, +1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0],
    [0, 0, -1.73, 0, 0, 0, 0, 0, 0, 2.9929],
    [0, 0, +1.73, 0, 0, 0, 0, 0, 0, 2.9929],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]

matrix_x = [[] for x 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]

def run_experiment():
```

```
    adekvat = False
    odnorid = False

    global average_y
    global matrix
    global dispersion_b2
    global student_lst
    global d
    global q
    global m
    global f3
    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\tПеревірка"
              .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]))

        while not odnorid:
            print("Матриця планування експеременту:")
            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("")

            dispersion_y = [0.0 for x in range(N)]
            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 = Perevirku.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\t Проводимо експеремент
повторно")
    return adekvat


if __name__ == '__main__':
    start = time.time()
    cnt = 0
    adekvat = 0

    while (time.time() - start) <= 10:
        cnt += 1
```

```
        try:
            adekvat += run_experiment()
        except Exception:
            continue

    print(f'За 10 секунд експеремент був адекватним {adekvat} разів з {cnt}')
```

Результат виконання роботи:

```
Отримане рівняння регресії
48.830 + 1.077 * X1 + 4.640 * X2 + 2.135 * X3 + 7.218 * X1X2 + 0.818 * X1X3 + 2.312 * X2X3+ 6.400 * X1X2X3 + 8.101 * X11^2 + 1.000 * X22^2 + 8.402 * X33^2 = ŷ
    Перевірка
ŷ1 = 107821.751 ≈ 107822.900
ŷ2 = 147335.580 ≈ 147336.150
ŷ3 = 247113.183 ≈ 247113.900
ŷ4 = 326412.011 ≈ 326412.150
ŷ5 = 456144.032 ≈ 456144.400
ŷ6 = 592143.361 ≈ 592143.150
ŷ7 = 1118957.463 ≈ 1118957.400
ŷ8 = 1448333.542 ≈ 1448332.900
ŷ9 = -41548.623 ≈ -41549.864
ŷ10 = 1163783.842 ≈ 1163784.406
ŷ11 = 155285.567 ≈ 155284.730
ŷ12 = 949948.730 ≈ 949948.890
ŷ13 = 426503.215 ≈ 426502.208
ŷ14 = 679164.553 ≈ 679164.883
ŷ15 = 551419.395 ≈ 551419.400
Матриця планування експеременту:
      X1          X2          X3          X1X2        X1X3        X2X3        X1X2X3      X1X1        X2X2        X3X3        Yi ->
    10.000      25.000      50.000      250.000     500.000     1250.000    12500.000   100.000     625.000     2500.000    107824.400  107823.400  107825.400  107818.400
    10.000      25.000      65.000      250.000     650.000     1625.000    16250.000   100.000     625.000     4225.000    147333.900  147334.900  147337.900  147337.900
    10.000      65.000      50.000      650.000     500.000     3250.000    32500.000   100.000     4225.000    2500.000    247117.400  247110.400  247116.400  247111.400
    10.000      65.000      65.000      650.000     650.000     4225.000    42250.000   100.000     4225.000    4225.000    326413.900  326410.900  326414.900  326408.900
    50.000      25.000      50.000      1250.000    2500.000    1250.000    62500.000   2500.000    625.000     2500.000    456145.400  456142.400  456142.400  456147.400
    50.000      25.000      65.000      1250.000    3250.000    1625.000    81250.000   2500.000    625.000     4225.000    592144.900  592142.900  592139.900  592144.900
    50.000      65.000      50.000      3250.000    2500.000    3250.000    162500.000  2500.000    4225.000    2500.000    1118958.400 1118955.400 1118955.400 1118960.400
    50.000      65.000      65.000      3250.000    3250.000    4225.000    211250.000  2500.000    4225.000    4225.000    1448329.900 1448332.900 1448334.900 1448333.900
    -4.600      45.000      57.500      -207.000    -264.500    2587.500    -11902.500  21.160      2025.000    3306.250    -41553.614  -41547.614  -41552.614  -41545.614
    64.600      45.000      57.500      2907.000    3714.500    2587.500    167152.500  4173.160    2025.000    3306.250    1163783.906 1163783.906 1163787.906 1163781.906
    30.000      10.400      57.500      312.000     1725.000    598.000     17940.000   900.000     108.160     3306.250    155281.980  155284.980  155283.980  155287.980
    30.000      79.600      57.500      2388.000    1725.000    4577.000    137310.000  900.000     6336.160    3306.250    949951.640  949946.640  949944.640  949952.640
    30.000      45.000      44.525      1350.000    1335.750    2003.625    60108.750   900.000     2025.000    1982.476    426502.708  426499.708  426498.708  426507.708
    30.000      45.000      70.475      1350.000    2114.250    3171.375    95141.250   900.000     2025.000    4966.726    679164.883  679167.883  679161.883  679164.883
    30.000      45.000      57.500      1350.000    1725.000    2587.500    77625.000   900.000     2025.000    3306.250    551420.650  551415.650  551421.650  551419.650
```

```
Критерій Кохрена:
Дисперсія однорідна при рівні значимості 0.05.
Отримане рівняння регресії з урахуванням критерія Стьюдента
48.830 + 1.077 * X1 + 4.640 * X2 + 2.135 * X3 + 7.218 * X1X2 + 0.818 * X1X3 + 2.312 * X2X3+ 6.400 * X1X2X3 + 8.101 * X11^2 + 1.000 * X22^2 + 8.402 * X33^2 = ŷ
    Перевірка
ŷ1 = 107821.751 ≈ 107822.900
ŷ2 = 147335.580 ≈ 147336.150
ŷ3 = 247113.183 ≈ 247113.900
ŷ4 = 326412.011 ≈ 326412.150
ŷ5 = 456144.032 ≈ 456144.400
ŷ6 = 592143.361 ≈ 592143.150
ŷ7 = 1118957.463 ≈ 1118957.400
ŷ8 = 1448333.542 ≈ 1448332.900
ŷ9 = -41548.623 ≈ -41549.864
ŷ10 = 1163783.842 ≈ 1163784.406
ŷ11 = 155285.567 ≈ 155284.730
ŷ12 = 949948.730 ≈ 949948.890
ŷ13 = 426503.215 ≈ 426502.208
ŷ14 = 679164.553 ≈ 679164.883
ŷ15 = 551419.395 ≈ 551419.400
Критерій Фішера
Рівняння регресії адекватне  оригіналу
За 10 секунд експеремент був адекватним 1874 разів з 1876
```