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

ЛАБОРАТОРНА РОБОТА №5

з дисципліни «Методи оптимізації та планування експерименту» на тему:

«ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ ПРИ ВИКОРИСТАННІ РІВНЯННЯ РЕГРЕСІЇ З УРАХУВАННЯМ КВАДРАТИЧНИХ ЧЛЕНІВ (ЦЕНТРАЛЬНИЙ ОРТОГОНАЛЬНИЙ КОМПОЗИЦІЙНИЙ ПЛАН)»

Виконав:
студент II курсу ФІОТ
групи ІВ-81 :
Бухтій О. В.
Перевірив:
Регіда П. Г.

Київ-2020

ЛАБОРАТОРНА РОБОТА №5.

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

Завдання

	x_1		x_2		x_3	
	min	max	min	max	min	max
107	-9	7	-4	7	-10	5

Y_{max} = 206.33(3)
Y_{min} = 192.33(3)

Лістинг програми

```
from prettytable import PrettyTable as prtt
from scipy.stats import f, t
import numpy as np
import functools as ft
import itertools as itr
from math import sqrt

# ~ Дано----------------------------
x1_min = -9
x1_max = 7

x2_min = -4
x2_max = 7

x3_min = -10
x3_max = 5

x_max_list = [x1_max, x2_max, x3_max]
x_min_list = [x1_min, x2_min, x3_min]

k = len(x_max_list)
m = 3
p = 0.95
with_interractions_and_squares = True

y_max = 200 + sum(x_max_list)/len(x_max_list)
y_min = 200 + sum(x_min_list)/len(x_min_list)
# ~ -------------------------------

def fisher_critical(prob, f4, f3):
    return f.ppf(p, f4, f3)
```

```
def student_critical(q, f3):
    return t.ppf((1 + (1 - q)) / 2, f3)

def cochran_critical(q, f1, f2):
    return 1 / (1 + (f2 - 1) / f.ppf(1 - q/f2, f1, (f2 - 1)*f1) )

def print_matrix(x_max_list, k, x_matr, y_matr, y_avg, y_disp):
      adequate_table  = prtt()
      group = ['x'+str(n+1) for n in range(len(x_max_list))]
      if with_interractions_and_squares :
            group_quantity = k+1
      else:
            group_quantity = 1
      header = []
      for j in range(group_quantity):
            header.extend(list(itr.combinations(group,j+1)))
      print(header)
      header.extend([('x'+str(n+1)+'^2',) for n in range(len(x_max_list))])
      print(['x'+str(n+1)+'^2' for n in range(len(x_max_list))])
      for i in range(len(y_matr[0])):
            header.append('y'+str(i+1))
      header.append('y_avg')
      header.append('s2{y}') # ~ variance or disp
      print(header)
      # ~ add squares
      final_matrix = np.hstack((x_matr,y_matr,[[round(avg,2)] for avg in y_avg],
[[round(disp,2)] for disp in y_disp]))
      header = list([ft.reduce(lambda x,y : x+y, i) for i in header])
      adequate_table.field_names = header
      for row_number in range(len(final_matrix)):
            adequate_table.add_row(final_matrix[row_number])
      print(adequate_table)

def print_equation(x_max_list, k, beta_list, ):
      group = ['x'+str(n+1) for n in range(len(x_max_list))]
      if with_interractions_and_squares:
            group_quantity = k+1
            header = []
            for j in range(group_quantity):
                  header.extend(list(itr.combinations(group,j+1)))
            header.extend([('x'+str(n+1)+'^2',) for n in
range(len(x_max_list))])
            x_names = list([ft.reduce(lambda x,y : x+y, i) for i in header])
            print(x_names)
            form = '*{:.4f} + '.join(x_names)+'*{:.4f} =  y'
      else:
            group_quantity = 1
            header = []
            for j in range(group_quantity):
                  header.extend(list(itr.combinations(group,j+1)))
            x_names = list([ft.reduce(lambda x,y : x+y, i) for i in header])
            form = '{:.4f} + '+'*{:.4f} + '.join(x_names)+'*{:.4f} =  y'
      print(form.format(*beta_list))

adequacy = False
while adequacy == False :
      # ~ Normalized matrix
      # ~ factors
      items = [(-1,1) for i in range(k)]
      f_matrix = list(itr.product(*items))
      # ~ add zor t
      initial_f_matrix = f_matrix.copy()
```

```
      l = 1.215
      zor_t_list = []
      for i in range(k):
            zor_t_row_pos = tuple([ l if i==j else 0 for j in range(k)])
            zor_t_row_neg = tuple([ -l if i==j else 0 for j in range(k)])
            zor_t_list.append(zor_t_row_pos)
            zor_t_list.append(zor_t_row_neg)
      zero_t_row = tuple([0 for j in range(k)])
      zor_t_list.append(zero_t_row)
      f_matrix.extend(zor_t_list)
      # ~ with interractions--------------------------
      if with_interractions_and_squares :
            interractions_matrix1 = []
            group_quantity = k - 1
            for i in f_matrix:
                  comb_list = []
                  for j in range(group_quantity):
                        comb_list.extend(list(itr.combinations(i,j+2)))
                  comb_values = [round(np.prod(k),2) for k in comb_list]
                  squared_values = [round(x**2,2) for x in i]
                  comb_and_squared_values = []
                  comb_and_squared_values.extend(comb_values)
                  comb_and_squared_values.extend(squared_values)
                  interractions_matrix1.append(comb_and_squared_values)
      # ~ ---------------------------------------------
      if with_interractions_and_squares :
            norm_matrix = np.hstack((f_matrix, interractions_matrix1))
#x0_vector,
      else:
            norm_matrix = np.hstack((f_matrix)) #x0_vector,
      # ~ Naturalized matrix
      col_list = []
      for i in range(len(f_matrix[0])):
            col1 = [row[i] for row in initial_f_matrix]
            col1 = list(map(lambda x : x_max_list[i] if x==1 else x_min_list[i],
col1))

            col2 = [row[i] for row in zor_t_list]
            x0i =(x_max_list[i]+x_min_list[i])/2
            deltaxi = x_max_list[i]-x0i
            col2 = list(map(lambda x : round(x0i,2) if x==0 else
round(x*deltaxi+x0i,2), col2))

            col12 = col1+col2
            col_list.append(col12)
      print(col_list)
      nf_matrix = list(zip(*col_list)) # ~ naturalized factors

      # ~ with interractions--------------------------
      if with_interractions_and_squares :
            print("here")
            interractions_matrix2 = []
            group_quantity = k - 1
            for i in nf_matrix:
                  comb_list = []
                  for j in range(group_quantity):
                        comb_list.extend(list(itr.combinations(i,j+2)))
                  comb_values = [round(np.prod(k),2) for k in comb_list]
                  squared_values = [round(x**2,2) for x in i]
                  comb_and_squared_values = []
                  comb_and_squared_values.extend(comb_values)
                  comb_and_squared_values.extend(squared_values)
```

```
            interractions_matrix2.append(comb_and_squared_values)
    # ~ ---------------------------------------------
    if with_interractions_and_squares :
        natur_matrix = np.hstack((nf_matrix, interractions_matrix2))
#x0_vector,
    else:
        natur_matrix = np.hstack((nf_matrix)) #x0_vector,
    # ~
--------------------------------------------------------------------------------
    escape = False
    while escape == False:
        y_matr = np.random.randint(y_min, y_max,(len(natur_matrix),m))
        y_avg = [sum(i)/len(i) for i in y_matr]
        y_disp = [] # ~ i.e. variance
        for i in range(len(natur_matrix)):
            tmp_disp = 0
            for j in range(m):
                tmp_disp += ((y_matr[i][j] - y_avg[i]) ** 2) / m
            y_disp.append(tmp_disp)
        f1 = m - 1
        f2 = len(natur_matrix)
        q = 1 - p
    # ~ Cochran test
        Gp = max(y_disp) / sum(y_disp)
        Gt = cochran_critical(q, f1, f2)
        # ~ print(Gt, Gp)
        if Gt > Gp:
            escape = True
            form = 'Cochran`s test passed with significance level {:.4f} :
Gt > Gp'
        else:
            m += 1
            form = 'Cochran`s test failed with significance level {:.4f} :
Gt < Gp'
        print(form.format(q))
        print('Gt = {}\nGp = {}'.format(Gt, Gp))

    # ~
--------------------------------------------------------------------------------
    if with_interractions_and_squares :
        beta_list_norm1 = list(np.linalg.solve(norm_matrix[3:13],
y_avg[3:13]))
        beta_list_natur1 = list(np.linalg.solve(natur_matrix[3:13],
y_avg[3:13]))
        beta_list_norm2 = list(np.linalg.solve(norm_matrix[1:11],
y_avg[1:11]))
        beta_list_natur2 = list(np.linalg.solve(natur_matrix[1:11],
y_avg[1:11]))
        beta_list_norm = list(zip(beta_list_norm1,beta_list_norm2))
        beta_list_norm =list(map(lambda x : sum(x)/2, beta_list_norm))
        beta_list_natur = list(zip(beta_list_natur1,beta_list_natur2))
        beta_list_natur =list(map(lambda x : sum(x)/2, beta_list_natur))

    print_matrix(x_max_list, k, norm_matrix, y_matr, y_avg, y_disp)
    print('Equation with normalized coefficients : ')
    print_equation(x_max_list, k, beta_list_norm)
    print_matrix(x_max_list, k, natur_matrix, y_matr, y_avg, y_disp)
    print('Equation with naturalized coefficients : ')
    print_equation(x_max_list, k, beta_list_natur)
    print(beta_list_natur)
    # ~
--------------------------------------------------------------------------------
```

```
    print('\nStudent`s test')
    N = len(beta_list_norm)
    f3 = f1 * f2
    S2b = sum(y_disp)**2 / (N * N * m)
    Sb = sqrt(S2b)
    betast_list = []
    for i in range(len(norm_matrix[0])):
        x_list_tmp = norm_matrix[:,i]
        beta_tmp = ((sum([np.prod(i) for i in list(zip(x_list_tmp,
y_avg))])) / N)
        betast_list.append(beta_tmp)

    T_list = [abs(beta)/Sb for beta in betast_list]
    T = student_critical(q, f3)
    print('T = '+str(T)+ '\nT_list = '+str(list(map(lambda x : round(x, 2),
T_list))))
    for i in range(len(T_list)):
        if T_list[i] < T :
            T_list[i] = 0
            beta_list_natur[i] = 0
    print('Fixed beta_list = '+ str(list(map(lambda x :round(x, 2),
beta_list_natur))))
    print('Equation without insignificant coefficients : ')
    print_equation(x_max_list, k, beta_list_natur)

    print("\nFisher test")
    equation_y_list = []
    for i in range(len(natur_matrix[0])):
        x_list_tmp = natur_matrix[i]
        y_tmp = sum([np.prod(i) for i in list(zip(x_list_tmp,
beta_list_natur))])
        equation_y_list.append(y_tmp)

    beta_list_natur = list(filter(lambda i : (i != 0), beta_list_natur))
    d = len(beta_list_natur)
    f4 = N - d
    S2ad = m * (sum([(i[0]-i[1])**2 for i in list(zip(equation_y_list,
y_avg))])) /f4
    Fp = sqrt(S2ad) / S2b
    Ft = fisher_critical(p, f4, f3)
    print('Fp = '+ str(Fp)+"\nFt = "+str(Ft))
    if Fp > Ft:
        print("     The regression equation is inadequate at the
significance level {:.2f}".format(q))
        with_interractions_and_squares = True
    else:
        print("     The regression equation is adequate at the significance
level {:.2f}".format(q))
        adequacy = True
```

Результат виконання

```
Equation with normalized coefficients :
x1*1.2346 + x2*0.9160 + x3*123.0028 + x1x2*-0.6660 + x1x3*-123.9194 + x2x3*-123.7675 + x1x2x3*122.0175 + x1^2*132.3198 + x2^2*132.8066 + x3^2*-67.6110 =  y
+--------+-------+--------+--------+-------+--------+--------+--------+-------+--------+-------+-------+-------+--------+-------+
|   x1   |   x2  |   x3   |  x1x2  |  x1x3 |  x2x3  | x1x2x3 |  x1^2  |  x2^2 |  x3^2  |   y1  |   y2  |   y3  | y_avg  | s2{y} |
+--------+-------+--------+--------+-------+--------+--------+--------+-------+--------+-------+-------+-------+--------+-------+
|  -9.0  |  -4.0 | -10.0  |  36.0  |  90.0 |  40.0  | -360.0 |  81.0  |  16.0 | 100.0  | 198.0 | 202.0 | 195.0 | 198.33 |  8.22 |
|  -9.0  |  -4.0 |  5.0   |  36.0  | -45.0 | -20.0  | 180.0  |  81.0  |  16.0 |  25.0  | 193.0 | 193.0 | 204.0 | 196.67 | 26.89 |
|  -9.0  |  7.0  | -10.0  | -63.0  |  90.0 | -70.0  | 630.0  |  81.0  |  49.0 | 100.0  | 200.0 | 196.0 | 192.0 | 196.0  | 10.67 |
|  -9.0  |  7.0  |  5.0   | -63.0  | -45.0 |  35.0  | -315.0 |  81.0  |  49.0 |  25.0  | 198.0 | 204.0 | 195.0 | 199.0  |  14.0 |
|  7.0   |  -4.0 | -10.0  | -28.0  | -70.0 |  40.0  | 280.0  |  49.0  |  16.0 | 100.0  | 195.0 | 198.0 | 200.0 | 197.67 |  4.22 |
|  7.0   |  -4.0 |  5.0   | -28.0  |  35.0 | -20.0  | -140.0 |  49.0  |  16.0 |  25.0  | 196.0 | 198.0 | 204.0 | 199.33 | 11.56 |
|  7.0   |  7.0  | -10.0  |  49.0  | -70.0 | -70.0  | -490.0 |  49.0  |  49.0 | 100.0  | 204.0 | 203.0 | 198.0 | 201.67 |  6.89 |
|  7.0   |  7.0  |  5.0   |  49.0  |  35.0 |  35.0  | 245.0  |  49.0  |  49.0 |  25.0  | 192.0 | 192.0 | 205.0 | 196.33 | 37.56 |
|  8.72  |  1.5  |  -2.5  | 13.08  | -21.8 | -3.75  | -32.7  | 76.04  |  2.25 |  6.25  | 194.0 | 194.0 | 204.0 | 197.33 | 22.22 |
| -10.72 |  1.5  |  -2.5  | -16.08 |  26.8 | -3.75  |  40.2  | 114.92 |  2.25 |  6.25  | 195.0 | 195.0 | 193.0 | 194.33 |  0.89 |
|  -1.0  |  8.18 |  -2.5  | -8.18  |  2.5  | -20.45 | 20.45  |  1.0   | 66.91 |  6.25  | 195.0 | 203.0 | 195.0 | 197.67 | 14.22 |
|  -1.0  | -5.18 |  -2.5  |  5.18  |  2.5  | 12.95  | -12.95 |  1.0   | 26.83 |  6.25  | 195.0 | 192.0 | 198.0 | 195.0  |  6.0  |
|  -1.0  |  1.5  |  6.61  |  -1.5  | -6.61 |  9.92  | -9.92  |  1.0   |  2.25 | 43.69  | 195.0 | 197.0 | 204.0 | 198.67 | 14.89 |
|  -1.0  |  1.5  | -11.61 |  -1.5  | 11.61 | -17.42 | 17.42  |  1.0   |  2.25 | 134.79 | 194.0 | 192.0 | 204.0 | 196.67 | 27.56 |
|  -1.0  |  1.5  |  -2.5  |  -1.5  |  2.5  | -3.75  |  3.75  |  1.0   |  2.25 |  6.25  | 199.0 | 202.0 | 202.0 | 201.0  |  2.0  |
+--------+-------+--------+--------+-------+--------+--------+--------+-------+--------+-------+-------+-------+--------+-------+
Equation with naturalized coefficients :
x1*-6.4416 + x2*-32.3208 + x3*26.6530 + x1x2*1.8009 + x1x3*-5.1166 + x2x3*-5.1273 + x1x2x3*0.7264 + x1^2*3.0864 + x2^2*6.5513 + x3^2*-1.8209 =  y

Student`s test
T = 2.0422724563012373
T_list = [0.07, 0.04, 0.0, 0.01, 0.04, 0.02, 0.1, 18.04, 18.06, 18.09]
Fixed beta_list = [0, 0, 0, 0, 0, 0, 0, 3.09, 6.55, -1.82]
Equation without insignificant coefficients :
x1*0.0000 + x2*0.0000 + x3*0.0000 + x1x2*0.0000 + x1x3*0.0000 + x2x3*0.0000 + x1x2x3*0.0000 + x1^2*3.0864 + x2^2*6.5513 + x3^2*-1.8209 =  y

Fisher test
Fp = 2.328352915185639
Ft = 2.3343439648447806
        The regression equation is adequate at the significance level 0.05
```

Висновки

Під час виконання лабораторної роботи було реалізовано завдання . Отримані результати збігаються , отже, експеримент було поставлено правильно.