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 №6

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

«ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ ПРИ ВИКОРИСТАННІ РІВНЯННЯ РЕГРЕСІЇ З КВАДРАТИЧНИМИ ЧЛЕНАМИ»

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

Київ-2020

ЛАБОРАТОРНА РОБОТА №6.

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

Завдання

	x_1		x_2		x_3	
	min	max	min	max	min	max
107	-5	15	-15	35	15	30

y=3.9+5.6*x1+7.9*x2+7.3*x3+2.0*x1*x1+0.5*x2*x2+4.2*x3*x3+1.5*x1*x2+0.1*x1*x3+9.9*x2*x3+5.3*x1*x2*x3 + random.random()*10 - 5

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
import random

# ~ Дано----------------------------
x1_min = -5
x1_max = 15

x2_min = -15
x2_max = 35

x3_min = 15
x3_max = 30

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

k = len(x_max_list)
m = 2
p = 0.95
with_interractions_and_squares = False

#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)))
      if with_interractions_and_squares:
            header.extend([('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
      # ~ 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])
            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} + '.join(x_names)+'*{:.4f} =  y'
      print(form.format(*beta_list))

def generate_y_matr(natur_matrix,m):
      y_matr = []
      natur_list = list(map(lambda x: list(x),natur_matrix))
      for i in natur_list:
            y_row = []
            for j in range(m):
                  x1 = i[0]
                  x2 = i[1]
                  x3 = i[2]
                  y =
3.9+5.6*x1+7.9*x2+7.3*x3+2.0*x1*x1+0.5*x2*x2+4.2*x3*x3+1.5*x1*x2+0.1*x1*x3+9.9*x
2*x3+5.3*x1*x2*x3 + random.random()*10 - 5
```

```
                  y_row.append(round(y,3))
            y_matr.append(y_row)
      return y_matr

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 = round(sqrt(len(x_max_list)),2)
      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 = 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)
      nf_matrix = list(zip(*col_list)) # ~ naturalized factors

      # ~ with interractions--------------------------
      if with_interractions_and_squares :
            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 = nf_matrix #x0_vector,
      # ~
----------------------------------------------------------------------------
      escape = False
      while escape == False:
            y_matr = generate_y_matr(natur_matrix, m) #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))
      else:
            beta_list_norm = list(np.linalg.solve(norm_matrix[1:4], y_avg[1:4]))
            beta_list_natur = list(np.linalg.solve(natur_matrix[1:4],
y_avg[1:4]))
      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('\nStudent`s test')
      N = len(norm_matrix)
      f3 = f1 * f2
      S2b = sum(y_disp)**2 / (N * N * m)
      Sb = S2b
      betast_list = []
      for i in range(len(norm_matrix[0])):
            x_list_tmp = np.array((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)):
            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

      print('\nResult check')
      counter = 1
      for i in list(zip(equation_y_list, y_avg)):
            print('y{counter:2} = {:13.4f} ;  y_avg{counter:2} =
{:13.4f} ;'.format(i[0], i[1], counter = counter))
            counter += 1
```

Результат виконання

```
+-------+--------+-------+---------+---------+---------+----------+--------+---------+---------+------------+------------+----------+-------+
|   x1  |   x2   |   x3  |   x1x2  |   x1x3  |   x2x3  |  x1x2x3  |  x1^2  |   x2^2  |   x3^2  |     y1     |     y2     |   y_avg  | s2{y} |
+-------+--------+-------+---------+---------+---------+----------+--------+---------+---------+------------+------------+----------+-------+
|  -5.0 | -15.0  |  15.0 |   75.0  |  -75.0  |  -225.0 |  1125.0  |  25.0  |  225.0  |  225.0  |  4913.467  |  4911.123  |   4912.3 |  1.37 |
|  -5.0 | -15.0  |  30.0 |   75.0  |  -150.0 |  -450.0 |  2250.0  |  25.0  |  225.0  |  900.0  | 11590.258  | 11584.534  |  11587.4 |  8.19 |
|  -5.0 |  35.0  |  15.0 |  -175.0 |  -75.0  |  525.0  | -2625.0  |  25.0  |  1225.0 |  225.0  | -7013.045  | -7018.294  | -7015.67 |  6.89 |
|  -5.0 |  35.0  |  30.0 |  -175.0 |  -150.0 |  1050.0 | -5250.0  |  25.0  |  1225.0 |  900.0  | -12796.883 | -12789.591 | -12793.24 | 13.29 |
|  15.0 | -15.0  |  15.0 |  -225.0 |  225.0  |  -225.0 | -3375.0  | 225.0  |  225.0  |  225.0  | -18848.045 | -18840.217 | -18844.13 | 15.32 |
|  15.0 | -15.0  |  30.0 |  -225.0 |  450.0  |  -450.0 | -6750.0  | 225.0  |  225.0  |  900.0  | -35988.535 | -35987.628 | -35988.08 |  0.21 |
|  15.0 |  35.0  |  15.0 |  525.0  |  225.0  |  525.0  |  7875.0  | 225.0  |  1225.0 |  225.0  |  50226.86  |  50226.024 |  50226.44 |  0.17 |
|  15.0 |  35.0  |  30.0 |  525.0  |  450.0  |  1050.0 | 15750.0  | 225.0  |  1225.0 |  900.0  | 100131.056 | 100123.706 | 100127.38 | 13.51 |
|  22.3 |  10.0  |  22.5 |  223.0  |  501.75 |  225.0  |  5017.5  | 497.29 |  100.0  |  506.25 | 32745.486  | 32752.766  |  32749.13 | 13.25 |
| -12.3 |  10.0  |  22.5 |  -123.0 | -276.75 |  225.0  | -2767.5  | 151.29 |  100.0  |  506.25 | -9997.677  | -9990.627  |  -9994.15 | 12.43 |
|  5.0  | 53.25  |  22.5 |  266.25 |  112.5  | 1198.12 | 5990.62  |  25.0  | 2835.56 |  506.25 | 48231.337  | 48237.709  |  48234.52 | 10.15 |
|  5.0  | -33.25 |  22.5 | -166.25 |  112.5  | -748.12 | -3740.62 |  25.0  | 1105.56 |  506.25 | -24803.29  | -24803.933 | -24803.61 |  0.1  |
|  5.0  |  10.0  | 35.48 |   50.0  |  177.4  |  354.8  |  1774.0  |  25.0  |  100.0  | 1258.83 | 18759.702  | 18761.465  |  18760.58 |  0.78 |
|  5.0  |  10.0  |  9.53 |   50.0  |  47.65  |   95.3  |  476.5   |  25.0  |  100.0  |  90.82  |  4214.791  |  4209.879  |  4212.34  |  6.03 |
+-------+--------+-------+---------+---------+---------+----------+--------+---------+---------+------------+------------+----------+-------+
Equation with naturalized coefficients :
x1*5.1433 + x2*7.7715 + x3*7.7947 + x1x2*1.5143 + x1x3*0.1191 + x2x3*9.9036 + x1x2x3*5.2993 + x1^2*2.0045 + x2^2*0.5014 + x3^2*4.1864 =  y

Student`s test
T = 2.1447866879169273
T_list = [467.83, 799.41, 159.28, 653.9, 86.27, 147.82, 215.26, 433.91, 439.39, 435.68]
Fixed beta_list = [5.14, 7.77, 7.79, 1.51, 0.12, 9.9, 5.3, 2.0, 0.5, 4.19]
Equation without insignificant coefficients :
x1*5.1433 + x2*7.7715 + x3*7.7947 + x1x2*1.5143 + x1x3*0.1191 + x2x3*9.9036 + x1x2x3*5.2993 + x1^2*2.0045 + x2^2*0.5014 + x3^2*4.1864 =  y

Fisher test
Fp = 0.19096454432834317
Ft = 3.112249847961388
       The regression equation is adequate at the significance level 0.05

Result check
y 1 =     4917.5199 ;  y_avg 1 =     4912.2950 ;
y 2 =    11584.6960 ;  y_avg 2 =    11587.3960 ;
y 3 =    -7015.7568 ;  y_avg 3 =    -7015.6695 ;
y 4 =   -12793.2370 ;  y_avg 4 =   -12793.2370 ;
y 5 =   -18844.1310 ;  y_avg 5 =   -18844.1310 ;
y 6 =   -35988.0815 ;  y_avg 6 =   -35988.0815 ;
y 7 =    50226.4420 ;  y_avg 7 =    50226.4420 ;
y 8 =   100127.3810 ;  y_avg 8 =   100127.3810 ;
y 9 =    32749.1260 ;  y_avg 9 =    32749.1260 ;
y10 =    -9994.1520 ;  y_avg10 =    -9994.1520 ;
y11 =    48234.5230 ;  y_avg11 =    48234.5230 ;
y12 =   -24803.6871 ;  y_avg12 =   -24803.6115 ;
y13 =    18761.7770 ;  y_avg13 =    18760.5835 ;
y14 =     4208.4976 ;  y_avg14 =     4212.3350 ;
```

Висновки

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