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 № 3**

з дисципліни «Методи оптимізації та планування експерименту» на тему

ВИКОНАВ:

студент ІІ курсу

групи ІО-93

Губенко Владислав

ПЕРЕВІРИВ:

Регіда П. Г.

Київ – 2021

|  |  |  |
| --- | --- | --- |
|  | Max | Min |
| x1 | 30 | -15 |
| x2 | 65 | 25 |
| x3 | -5 | -15 |

**Лістинг**

from random import \*  
import numpy as np  
from numpy.linalg import solve  
from scipy.stats import f, t  
from functools import partial  
  
  
class FractionalExperiment:  
  
 def \_\_init\_\_(self, n, m):  
 self.n = n  
 self.m = m  
 self.x\_min = (-30 + -20 + -30) / 3  
 self.x\_max = (20 + 40 + -15) / 3  
 self.y\_max = round(200 + self.x\_max)  
 self.y\_min = round(200 + self.x\_min)  
 self.x\_norm = [[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]]  
 self.x\_range = [(-30, 20), (-20, 40), (-30, -15)]  
 self.y = np.zeros(shape=(self.n, self.m))  
 self.y\_new = []  
 for i in range(self.n):  
 for j in range(self.m):  
 self.y[i][j] = randint(self.y\_min, self.y\_max)  
 self.y\_av = [round(sum(i) / len(i), 2) for i in self.y]  
 self.x\_norm = self.x\_norm[:len(self.y)]  
 self.x = np.ones(shape=(len(self.x\_norm), len(self.x\_norm[0])))  
 for i in range(len(self.x\_norm)):  
 for j in range(1, len(self.x\_norm[i])):  
 if self.x\_norm[i][j] == -1:  
 self.x[i][j] = self.x\_range[j - 1][0]  
 else:  
 self.x[i][j] = self.x\_range[j - 1][1]  
 self.f1 = m - 1  
 self.f2 = n  
 self.f3 = self.f1 \* self.f2  
 self.q = 0.05  
  
 def regression(self, x, b):  
  
 y = sum([x[i] \* b[i] for i in range(len(x))])  
 return y  
  
 def count\_koefs(self):  
  
 mx1 = sum(self.x[:, 1]) / self.n  
 mx2 = sum(self.x[:, 2]) / self.n  
 mx3 = sum(self.x[:, 3]) / self.n  
 my = sum(self.y\_av) / self.n  
 a12 = sum([self.x[i][1] \* self.x[i][2] for i in range(len(self.x))]) / self.n  
 a13 = sum([self.x[i][1] \* self.x[i][3] for i in range(len(self.x))]) / self.n  
 a23 = sum([self.x[i][2] \* self.x[i][3] for i in range(len(self.x))]) / self.n  
 a11 = sum([i \*\* 2 for i in self.x[:, 1]]) / self.n  
 a22 = sum([i \*\* 2 for i in self.x[:, 2]]) / self.n  
 a33 = sum([i \*\* 2 for i in self.x[:, 3]]) / self.n  
 a1 = sum([self.y\_av[i] \* self.x[i][1] for i in range(len(self.x))]) / self.n  
 a2 = sum([self.y\_av[i] \* self.x[i][2] for i in range(len(self.x))]) / self.n  
 a3 = sum([self.y\_av[i] \* self.x[i][3] for i in range(len(self.x))]) / self.n  
  
 X = [[1, mx1, mx2, mx3], [mx1, a11, a12, a13], [mx2, a12, a22, a23], [mx3, a13, a23, a33]]  
 Y = [my, a1, a2, a3]  
 B = [round(i, 2) for i in solve(X, Y)]  
 print('\nРівняння регресії')  
 print(f'y = {B[0]} + {B[1]}\*x1 + {B[2]}\*x2 + {B[3]}\*x3')  
  
 return B  
  
 def dispersion(self):  
  
 res = []  
 for i in range(self.n):  
 s = sum([(self.y\_av[i] - self.y[i][j]) \*\* 2 for j in range(self.m)]) / self.m  
 res.append(s)  
 return res  
  
 def kohren(self):  
  
 q1 = self.q / self.f1  
 fisher\_value = f.ppf(q=1 - q1, dfn=self.f2, dfd=(self.f1 - 1) \* self.f2)  
 G\_cr = fisher\_value / (fisher\_value + self.f1 - 1)  
 s = self.dispersion()  
 Gp = max(s) / sum(s)  
 return Gp, G\_cr  
  
 def student(self):  
  
 def bs():  
 res = [sum(1 \* y for y in self.y\_av) / self.n]  
 for i in range(3): # 4 - ксть факторів  
 b = sum(j[0] \* j[1] for j in zip(self.x[:, i], self.y\_av)) / self.n  
 res.append(b)  
 return res  
  
 S\_kv = self.dispersion()  
 s\_kv\_aver = sum(S\_kv) / self.n  
  
 # статиcтична оцінка дисперсії  
 s\_Bs = (s\_kv\_aver / self.n / self.m) \*\* 0.5  
 Bs = bs()  
 ts = [abs(B) / s\_Bs for B in Bs]  
 return ts  
  
 def fisher(self, d):  
  
 S\_ad = self.m / (self.n - d) \* sum([(self.y\_new[i] - self.y\_av[i]) \*\* 2 for i in range(len(self.y))])  
 S\_kv = self.dispersion()  
 S\_kv\_aver = sum(S\_kv) / self.n  
 F\_p = S\_ad / S\_kv\_aver  
 return F\_p  
  
 def check(self):  
  
 student = partial(t.ppf, q=1 - 0.025)  
 t\_student = student(df=self.f3)  
  
 print('\nПеревірка за критерієм Кохрена')  
 Gp, G\_kr = self.kohren()  
 print(f'Gp = {Gp}')  
 if Gp < G\_kr:  
 print(f'З ймовірністю {1 - self.q} дисперсії однорідні.')  
 else:  
 print("Необхідно збільшити кількість дослідів")  
 self.m += 1  
 FractionalExperiment(self.n, self.m)  
  
 ts = self.student()  
 print('\nПеревірка значущості коефіцієнтів за критерієм Стьюдента')  
 print('Критерій Стьюдента:\n', ts)  
 res = [t for t in ts if t > t\_student]  
 B = self.count\_koefs()  
 final\_k = [B[ts.index(i)] for i in ts if i in res]  
 self.sig\_coefs\_n = len(final\_k[1:])  
 print('Коефіцієнти {} статистично незначущі, тому ми виключаємо їх з рівняння.'.format(  
 [i for i in B if i not in final\_k]))  
  
 for j in range(self.n):  
 self.y\_new.append(self.regression([self.x[j][ts.index(i)] for i in ts if i in res], final\_k))  
  
 print(f'\nЗначення "y" з коефіцієнтами {final\_k}')  
 print(self.y\_new)  
  
 d = len(res)  
 f4 = self.n - d  
 F\_p = self.fisher(d)  
  
 fisher = partial(f.ppf, q=1 - 0.05)  
 f\_t = fisher(dfn=f4, dfd=self.f3) # табличне знач  
 print('\nПеревірка адекватності за критерієм Фішера')  
 print('Fp =', F\_p)  
 print('F\_t =', f\_t)  
 if F\_p < f\_t:  
 print('Математична модель адекватна експериментальним даним')  
 else:  
 print('Математична модель не адекватна експериментальним даним')  
  
total\_sig\_coefs\_q = 0  
n = 100  
for \_ in range(n):  
 experiment = FractionalExperiment(7, 8)  
 experiment.check()  
 total\_sig\_coefs\_q += experiment.sig\_coefs\_n  
print("\nКількість значущих коефіцієнтів за {} виконань програми: {}".format(n, total\_sig\_coefs\_q))