НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені Ігоря Сікорського”

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА 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)

Виконала Дуплій Наталя Володимирівна

Факультет ІОТ,

Група ІВ-82

Залікова книжка № ІВ-8209

Київ – 2020 р.

**Варіант**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| №варіанта | X1 | | X2 | | X3 | |
| min | max | min | max | min | max |
| 208 | -5 | 6 | -7 | 9 | -5 | 3 |

**Короткі теоретичні відомості**

Рівняння регресії ПФЕ з урахуванням квадратичних членів для трьох факторів має вигляд:

Це рівняння повинне відображати з певною точністю значення функції відгуку для значень факторів, які знаходяться у певних межах, і зокрема, для точок плану:

**Результати підготовки та виконання роботи**

Напишемо рівняння регресії з ефектом взаємодії:

Інтервали значень :

Значення, відповідні кодованим:

Складаємо матрицю планування для ОЦКП і заповнюємо таблицю кодованими значеннями .

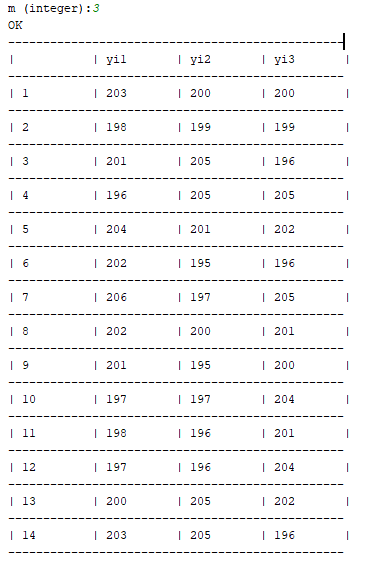
Обираємо m=3

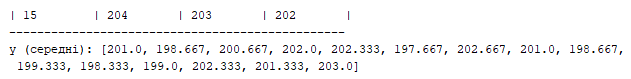
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **1** | -1 | -1 | -1 | 1 | 1 | 1 | -1 | 1 | 1 | 1 | 203 | 200 | 200 | 201 |
| **2** | -1 | -1 | 1 | 1 | -1 | -1 | 1 | 1 | 1 | 1 | 198 | 199 | 199 | 198.667 |
| **3** | -1 | 1 | -1 | -1 | 1 | -1 | 1 | 1 | 1 | 1 | 201 | 205 | 196 | 200.667 |
| **4** | -1 | 1 | 1 | -1 | -1 | 1 | -1 | 1 | 1 | 1 | 196 | 205 | 205 | 202 |
| **5** | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 | 1 | 1 | 204 | 201 | 202 | 202.333 |
| **6** | 1 | -1 | 1 | -1 | 1 | -1 | -1 | 1 | 1 | 1 | 202 | 195 | 196 | 197.667 |
| **7** | 1 | 1 | -1 | 1 | -1 | -1 | -1 | 1 | 1 | 1 | 206 | 197 | 205 | 202.667 |
| **8** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 202 | 200 | 201 | 201 |
| **9** | -1.215 | 0 | 0 | 0 | 0 | 0 | 0 | 1.47623 | 0 | 0 | 201 | 195 | 200 | 198.667 |
| **10** | 1.215 | 0 | 0 | 0 | 0 | 0 | 0 | 1.47623 | 0 | 0 | 197 | 197 | 204 | 199.333 |
| **11** | 0 | -1.215 | 0 | 0 | 0 | 0 | 0 | 0 | 1.47623 | 0 | 198 | 296 | 201 | 198.333 |
| **12** | 0 | 1.215 | 0 | 0 | 0 | 0 | 0 | 0 | 1.47623 | 0 | 197 | 196 | 204 | 199 |
| **13** | 0 | 0 | -1.215 | 0 | 0 | 0 | 0 | 0 | 0 | 1.47623 | 200 | 205 | 202 | 202.333 |
| **14** | 0 | 0 | 1.215 | 0 | 0 | 0 | 0 | 0 | 0 | 1.47623 | 203 | 205 | 196 | 201.333 |
| **15** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 204 | 203 | 202 | 203 |

Таблиця з натуралізованими значеннями

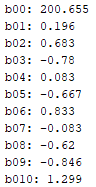
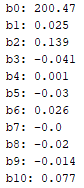
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **1** | -5 | -7 | -5 | 35 | 25 | 35 | -175 | 25 | 49 | 25 | 203 | 200 | 200 | 201 |
| **2** | -5 | -7 | 3 | 35 | -15 | -21 | 105 | 25 | 49 | 9 | 198 | 199 | 199 | 198.667 |
| **3** | -5 | 9 | -5 | -45 | 25 | -45 | 225 | 25 | 81 | 25 | 201 | 205 | 196 | 200.667 |
| **4** | -5 | 9 | 3 | -45 | -15 | 27 | -135 | 25 | 81 | 9 | 196 | 205 | 205 | 202 |
| **5** | 6 | -7 | -5 | -42 | -30 | 35 | 210 | 36 | 49 | 25 | 204 | 201 | 202 | 202.333 |
| **6** | 6 | -7 | 3 | -42 | 18 | -21 | -126 | 36 | 49 | 9 | 202 | 195 | 196 | 197.667 |
| **7** | 6 | 9 | -5 | 54 | -30 | -45 | -270 | 36 | 81 | 25 | 206 | 197 | 205 | 202.667 |
| **8** | 6 | 9 | 3 | 54 | 18 | 27 | 162 | 36 | 81 | 9 | 202 | 200 | 201 | 201 |
| **9** | -6.075 | 1 | -1 | -6.075 | 6.075 | -1 | 6.075 | 36.905625 | 1 | 1 | 201 | 195 | 200 | 198.667 |
| **10** | 7.29 | 1 | -1 | 7.29 | -7.29 | -1 | -7.29 | 53.1441 | 1 | 1 | 197 | 197 | 204 | 199.333 |
| **11** | 0.5 | -8.505 | -1 | -4.2525 | -0.5 | 8.505 | 4.2525 | 0.25 | 72.335025 | 1 | 198 | 296 | 201 | 198.333 |
| **12** | 0.5 | 10.935 | -1 | 5.4675 | -0.5 | -10.935 | -5.4675 | 0.25 | 119.574225 | 1 | 197 | 196 | 204 | 199 |
| **13** | 0.5 | 1 | -6.075 | 0.5 | -3.0375 | -6.075 | -3.0375 | 0.25 | 1 | 36.905625 | 200 | 205 | 202 | 202.333 |
| **14** | 0.5 | 1 | 3.645 | 0.5 | 1.8225 | 3.645 | 1.8225 | 0.25 | 1 | 13.286025 | 203 | 205 | 196 | 201.333 |
| **15** | 0.5 | 1 | -1 | 0.5 | -0.5 | -1 | -0.5 | 0.25 | 1 | 1 | 204 | 203 | 202 | 203 |

Проводимо експеримент у всіх точках плану



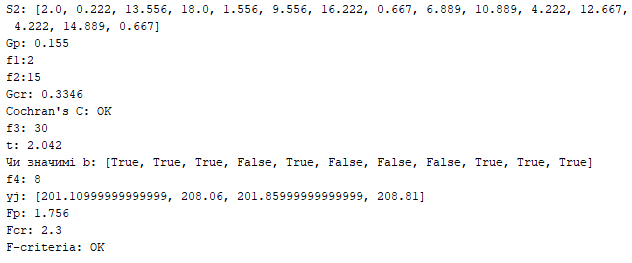


Середні значення функції відгуку та коефіцієнти рівняння регресії знайдені за допомогою програми.



Рівняння регресії:

Проводимо статистичні перевірки.



Дисперсія однорідна за критерієм Кохрена. Коефіцієнти b0, b1, b2, b4, b8, b9, b10 значимі, b3, b5, b6, b7 незначимі за критерієм Стьюдента. Модель адекватна за критерієм Фішера.

Скореговане рівняння регресії:

Перевірка рівнняння регресії:

**Висновки**

Проведено повний трьохфакторний експеримент з центральним ортогональним композиційним планом. Знайдено рівняння регресії адекватне об'єкту.

**Лістинг програми**

**import** random  
**import** numpy  
**import** copy  
**from** scipy.stats **import** t  
**from** scipy.stats **import** f  
**import** math  
  
**def** cochran(f1, f2, q):  
 fish = f.isf(q/f2, f1, (f2 - 1)\*f1)  
 result = fish/(fish + f2 - 1)  
 **return** result  
  
**def** print\_line(m):  
 print(**"-"** \* 12 \* (m+1))  
  
**def** coeffs\_criterias(yi, x, y):  
 **global** b, m  
 k = len(x[0])  
 mx = [[] **for** i **in** range(len(x) + 1)]  
 mx[0].append(k)  
 **for** i **in** range(1, len(x) + 1):  
 suma = round(sum(x[i-1]), 3)  
 mx[0].append(suma)  
 mx[i].append(suma)  
 **for** j **in** range(0, len(x)):  
 mx[i].append(round(sum([round(x[i-1][l] \* x[j][l], 3) **for** l **in** range(k)]), 3))  
  
 det = numpy.linalg.det(mx)  
 delta = round(det, 3)  
  
 my = [round(sum(yi), 5)]  
 **for** i **in** range(len(x)):  
 my.append(round(sum([yi[j]\*x[i][j] **for** j **in** range(k)]), 5))  
  
 b = [copy.deepcopy(mx) **for** i **in** range(len(x) + 1)]  
 **for** i **in** range(len(x) + 1):  
 **for** j **in** range(len(x) + 1):  
 b[i][j][i] = my[j]  
 b[i] = round(numpy.linalg.det(b[i])/delta, 3)  
 print(**"b"** + str(i) + **": "** + str(b[i]))  
  
 x0 = [[-1, -1, -1, -1, 1, 1, 1, 1, -1.215, 1.215, 0, 0, 0, 0, 0],  
 [-1, -1, 1, 1, -1, -1, 1, 1, 0, 0, -1.215, 1.215, 0, 0, 0],  
 [-1, 1, -1, 1, -1, 1, -1, 1, 0, 0, 0, 0, -1.215, 1.215, 0]]  
 x0.append([x0[0][i] \* x0[1][i] **for** i **in** range(k)])  
 x0.append([x0[0][i] \* x0[2][i] **for** i **in** range(k)])  
 x0.append([x0[1][i] \* x0[2][i] **for** i **in** range(k)])  
 x0.append([x0[0][i] \* x0[1][i] \* x0[2][i] **for** i **in** range(k)])  
 x0.append([round(x0[0][i] \*\* 2, 6) **for** i **in** range(k)])  
 x0.append([round(x0[1][i] \*\* 2, 6) **for** i **in** range(k)])  
 x0.append([round(x0[2][i] \*\* 2, 6) **for** i **in** range(k)])mx0 = [[0 **for** j **in** range(len(x0) + 1)] **for** i **in** range(len(x0) + 1)]  
  
 mx0[0][0] = k  
 **for** i **in** range(8, len(mx0)):  
 mx0[0][i] = round(sum(x0[i-1]), 3)  
 mx0[i][0] = mx0[0][i]  
 **for** j **in** range(8, len(mx0)):  
 mx0[i][j] = round(sum([x0[i-1][l]\*x0[j-1][l] **for** l **in** range(k)]), 3)  
 **for** i **in** range(1, 8):  
 mx0[i][i] = round(sum([x0[i-1][j]\*\*2 **for** j **in** range(k)]), 3)  
  
 det0 = numpy.linalg.det(mx0)  
  
 my0 = [round(sum(yi), 5)]  
 **for** i **in** range(len(x0)):  
 my0.append(round(sum([yi[j]\*x0[i][j] **for** j **in** range(k)]), 5))  
  
 b0 = [copy.deepcopy(mx0) **for** i **in** range(len(x0) + 1)]  
 **for** i **in** range(len(x0) + 1):  
 **for** j **in** range(len(x0) + 1):  
 b0[i][j][i] = my0[j]  
 b0[i] = round(numpy.linalg.det(b0[i])/det0, 3)  
 print(**"b0"** + str(i) + **": "** + str(b0[i]))  
  
 S2 = []  
 **for** i **in** range(len(y)):  
 S2.append(sum([(y[i][j] - yi[i])\*\*2 **for** j **in** range(len(y[i]))]))  
 S2[i] = round(S2[i]/len(y[i]), 3)  
 print(**"S2: "** + str(S2))  
  
 Gp = round(max(S2)/sum(S2), 3)  
 print(**"Gp: "** + str(Gp))  
  
 f1 = m - 1  
 f2 = k  
  
 print(**"f1:"** + str(f1))  
 print(**"f2:"** + str(f2))  
  
 alpha = 0.05  
  
 Gcr = round(cochran(f1, f2, alpha), 4)  
 print(**"Gcr: "** + str(Gcr))  
 **if** Gp < Gcr:  
 print(**"Cochran's C: OK"**)  
 **else**:  
 print(**"Cochran's C: :("**)  
 m += 1  
 **return** generate\_y(x)  
  
 S2v = sum(S2) / 4  
  
 S2b = round(S2v / (4 \* m), 3)  
 Sb = round(math.sqrt(S2b), 3)  
  
 f3 = f1 \* f2  
 print(**"f3: "** + str(f3))  
 tcr = round(t.ppf(1 - alpha / 2, df=f3), 3)  
 print(**"t: "** + str(tcr))  
 bs = []  
 ts = []  
 d = 0  
 bs.append(round(sum([yi[j] **for** j **in** range(len(yi))]) / len(yi), 3))  
 ts.append(round(bs[0] / Sb, 3))  
 **if** ts[0] > tcr:  
 ts[0] = **True** d += 1  
 **else**:  
 ts[0] = **False  
 for** i **in** range(len(x)):  
 bs.append(round(sum([yi[j] \* x[i][j] **for** j **in** range(len(yi))]) / len(yi), 3))  
 ts.append(round(bs[i+1] / Sb, 3))  
 **if** ts[i+1] > tcr:  
 ts[i+1] = **True** d += 1  
 **else**:  
 ts[i+1] = **False** print(**"Чи значимі b: "** + str(ts))  
  
 f4 = k - d  
 print(**"f4: "** + str(f4))  
 x = [[-30, -30, 0, 0],  
 [10, 60, 10, 60],  
 [10, 35, 35, 10]]  
 yj = []  
 **for** i **in** range(4):  
 yj.append(0)  
 **for** j **in** range(4):  
 **if** ts[j]:  
 **if** j == 0:  
 yj[i] += b[0]  
 **else**:  
 yj[i] += b[j] \* x[j - 1][i]  
 print(**"yj: "** + str(yj))  
  
 S2ad = round(m \* sum([(yj[i] - yi[i]) \*\* 2 **for** i **in** range(4)]) / f4, 3)  
  
 Fp = round(S2ad / S2v, 3)  
 print(**"Fp: "** + str(Fp))  
 Fcr = round(f.ppf(1 - alpha, f4, f3), 1)  
 print(**"Fcr: "** + str(Fcr))  
 **if** Fp < Fcr:  
 print(**"F-criteria: OK"**)  
 **else**:  
 print(**"F-criteria: :("**)  
  
**def** generate\_y(x):  
 k = len(x[0])  
 print\_line(m)  
 print(**"| "** + **'{:<10}'**.format(**""**), end=**""**)  
 **for** i **in** range(1, m+1):  
 print(**"| "** + **'{:<10}'**.format(**"yi"** + str(i)), end=**""**)  
 print(**"|"**)  
 print\_line(m)  
  
 y = []  
  
 **for** j **in** range(1, k+1):  
 y.append([])  
 print(**"| "** + **'{:<10}'**.format(j), end=**""**)  
 **for** i **in** range(1, m+1):  
 r = round(random.random() \* (max\_num - min\_num) + min\_num)  
 y[j-1].append(r)  
 print(**"| "** + **'{:<10}'**.format(r), end=**""**)  
 print(**"|"**)  
 print\_line(m)  
  
 yi = []  
 **for** i **in** range(k):  
 yi.append(round(1/m \* sum(y[i]), 3))  
  
 print(**"y (середні): "** + str(yi))  
  
 coeffs\_criterias(yi, x, y)  
  
max\_num = 206  
min\_num = 194.33  
x = [[-5, -5, -5, -5, 6, 6, 6, 6, -6.075, 7.29, 0.5, 0.5, 0.5, 0.5, 0.5],  
 [-7, -7, 9, 9, -7, -7, 9, 9, 1, 1, -8.505, 10.935, 1, 1, 1],  
 [-5, 3, -5, 3, -5, 3, -5, 3, -1, -1, -1, -1, -6.075, 3.645, -1]]  
  
k = len(x[0])  
x.append([x[0][i] \* x[1][i] **for** i **in** range(k)])  
x.append([x[0][i] \* x[2][i] **for** i **in** range(k)])  
x.append([x[1][i] \* x[2][i] **for** i **in** range(k)])  
x.append([x[0][i] \* x[1][i] \* x[2][i] **for** i **in** range(k)])  
x.append([round(x[0][i] \*\* 2, 6) **for** i **in** range(k)])  
x.append([round(x[1][i] \*\* 2, 6) **for** i **in** range(k)])  
x.append([round(x[2][i] \*\* 2, 6) **for** i **in** range(k)])  
**while True**:  
 m = input(**"m (integer):"**)  
 **if** m.isnumeric():  
 print(**"OK"**)  
 m = int(m)  
 **break  
 else**:  
 print(**"m must be integer"**)  
  
generate\_y(x)

**Результати виконання**

