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

Методи оптимізації та планування

Лабораторна робота №5

«Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів (центральний ортогональний композиційний план).»

Виконав:
студент групи ІО-82
Вербовський Ілля
Залікова книжка № 8205
Номер у списку групи 04
Перевірив ас. Регіда П. Г.

Київ 2020 р

Лістинг програми

```
import random, numpy
from scipy.stats import t,f


#test if dispersion is uniform using Kohren criteria
def kohren(mat_y, m, n):
    s = []
    for i in range(n):
        ks = 0
        for j in range(m):
            ks += (mat_y[i][-1] - mat_y[i][j]) ** 2
        s.append(ks / m)
    gp = max(s) / sum(s)
    fisher = table_fisher(0.95, n, m, 1)
    gt = fisher/(fisher+(m-1)-2)
    return gp < gt


#generate new experiment dat
def geny(n, m, y_max, y_min):
    mat_y = [[random.randint(y_min, y_max) for j in range(m)]for i in range(n)]
    for elem in mat_y:
        elem.append(sum(elem) / len(elem))
    return mat_y


def calcxi(n, listx):
    sumxi = 0
    for i in range(n):
        lsumxi = 1
        for j in range(len(listx)):
            lsumxi *= listx[j][i]
        sumxi += lsumxi
    return sumxi


#return all combinations of three array elements
def combination_mul(arr):
    return [1, *arr,
           arr[0]*arr[1],
           arr[0]*arr[2],
           arr[1]*arr[2],
           arr[0]*arr[1]*arr[2],
           arr[0]*arr[0],
           arr[1]*arr[1],
           arr[2]*arr[2]]


#calculate b in natural equation
def calcb(lmaty):
    a00 = [[],
           [xnatmod[0]],
           [xnatmod[1]],
           [xnatmod[2]],
           [xnatmod[0], xnatmod[1]],
           [xnatmod[0], xnatmod[2]],
           [xnatmod[1], xnatmod[2]],
           [xnatmod[0], xnatmod[1], xnatmod[2]],
           [xnatmod[0], xnatmod[0]],
           [xnatmod[1], xnatmod[1]],
           [xnatmod[2], xnatmod[2]]]
```

```
    a0 = [[calcxi(n, i + j) for j in a00] for i in a00]
    a = numpy.array(a0)
    c0 = [calcxi(n, [lmaty])]
    for i in range(len(a00) - 1):
        c0.append(calcxi(n, a00[i + 1] + [lmaty]))
    c = numpy.array(c0)
    b = numpy.linalg.solve(a, c)

    return b


#return table for Student criteria
def table_student(prob, n, m):
    x_vec = [i*0.0001 for i in range(int(5/0.0001))]
    par = 0.5 + prob/0.1*0.05
    f3 = (m - 1) * n
    for i in x_vec:
        if abs(t.cdf(i, f3) - par) < 0.000005:
            return i


#return table for Fisher criteria
def table_fisher(prob, n, m, d):
    x_vec = [i*0.001 for i in range(int(10/0.001))]
    f3 = (m - 1) * n
    for i in x_vec:
        if abs(f.cdf(i, n-d, f3)-prob) < 0.0001:
            return i


#test relevance of b using Student criteria
def student(n, m, mat_y):
    disp = []
    for i in mat_y:
        s = 0
        for k in range(m):
            s += (i[-1] - i[k]) ** 2
        disp.append(s / m)

    sbt = (sum(disp) / n / n / m) ** (0.5)

    bs = []
    for i in range(11):
        ar = []
        for j in range(len(mat_y)):
            ar.append(mat_y[j][-1] * combination_mul(xnorm[j])[i] / n)
        bs.append(sum(ar))

    t = [(bs[i] / sbt) for i in range(11)]
    tt = table_student(0.95, n, m)
    st = [i > tt for i in t]
    return st


#test adequativity of equation using Fisher criteria
def fisher(b_0, x_mod, n, m, d, mat_y):
    if d == n:
        return True
    disp = []
    for i in mat_y:
        s = 0
        for k in range(m):
            s += (i[-1] - i[k]) ** 2
```

```
        disp.append(s / m)

    sad = sum([(sum([combination_mul(xnat[i])[j] * b_0[j] for j in range(11)]) - mat_y[i][-1])
** 2 for i in range(n)])
    sad = sad * m / (n - d)
    fp = sad / sum(disp) / n
    ft = table_fisher(0.95, n, m, d)
    return fp < ft


def console_output():
    #natural=============================================================
    titles_x = ["№", "X1", "X2", "X3", "X1*X2", "X1*X3", "X2*X3", "X1*X2*X3", "X1^2", "X2^2",
"X3^2"]

    for j in range(11):
        s = ""
        if j == 0:
            s = "| {:2s} |"
        if j >= 1 and j < 4:
            s = " {:6s} |"
        if j >= 4 and j < 7:
            s = " {:7s} |"
        if j == 7:
            s = " {:8s} |"
        if j > 7 and j < 11:
            s = " {:6s} |"
        print(s.format(titles_x[j]), end="")

    for i in range(m):
        print(" Yi{:d} |".format(i+1), end="")
    print("   Ys   |   Ye   |", end="")

    print()
    for i in range(n):
        print("| {:2d} |".format(i), end="")
        for j in range(1, 11):
            s = ""
            if j >= 1 and j < 4:
                s = " {:6.3f} |"
            if j >= 4 and j < 7:
                s = " {:7.3f} |"
            if j == 7:
                s = " {:8.3f} |"
            if j > 7 and j < 11:
                s = " {:6.3f} |"
            print(s.format(combination_mul(xnat[i])[j]), end="")

        for j in maty[i][:-1]:
            print(" {:3d} |".format(j), end="")
        print(" {:6.2f} | {:6.2f} |"
              .format(maty[i][-1],
                      sum([combination_mul(xnat[i])[j] * b0[j] * d_arr[j] for j in
range(11)])), end="")

        print()

    print("\n\tNatural linear regrecy equation:  Y = ", end="")
    if d_arr[0] != 0:
        print("{:.5f}".format(b0[0]), end="")
    for i in range(1, 11):
        if d_arr[i] != 0:
            print(" + {:.5f}*{}".format(b0[i], titles_x[i]), end="")
    print()
```

```
x1min = -5
x1max = 5
x01 = (x1min + x1max) / 2
xl1 = 1.215*(x1max-x01)+x01

x2min = -5
x2max = 6
x02 = (x2min + x2max) / 2
xl2 = 1.215*(x2max-x02)+x02

x3min = -4
x3max = 8
x03 = (x3min + x3max) / 2
xl3 = 1.215*(x3max-x03)+x03

ymax = round(200 + (x1max + x2max + x3max) / 3)
ymin = round(200 + (x1min + x2min + x3min) / 3)

xnorm = [[-1, -1, -1],
         [-1, 1, 1],
         [1, -1, 1],
         [1, 1, -1],
         [-1, -1, 1],
         [-1, 1, -1],
         [1, -1, -1],
         [1, 1, 1],
         [-1.125, 0, 0],
         [1.125, 0, 0],
         [0, -1.125, 0],
         [0, 1.125, 0],
         [0, 0, -1.125],
         [0, 0, 1.125],
         [0, 0, 0]]

xnat = [[x1min, x2min, x3min],
        [x1min, x2min, x3max],
        [x1min, x2max, x3min],
        [x1min, x2max, x3max],
        [x1max, x2min, x3min],
        [x1max, x2min, x3max],
        [x1max, x2max, x3min],
        [x1max, x2max, x3max],
        [-xl1, x02, x03],
        [xl1, x02, x03],
        [x01, -xl2, x03],
        [x01, xl2, x03],
        [x01, x02, -xl3],
        [x01, x02, xl3],
        [x01, x02, x03]]

n = 15
m = 3

while True:
    while True:
        print("\nStart. Current m = {}\n".format(m))
        xnatmod = [[xnat[i][j] for i in range(15)] for j in range(3)]
        maty = geny(n, m, ymax, ymin)
        matymod = [maty[i][-1] for i in range(len(maty))]

        koh = kohren(maty, 3, 15)
        print("Dispersion uniform is {}, with probability = {:.2}".format(koh, 0.95))
```

```
        if koh:
            break
        else:
            m += 1

    b0 = calcb(matymod)

    d_arr = student(n, m, maty)
    d = sum(d_arr)

    fishercheck = fisher(b0, xnatmod, n, m, d, maty)
    print("Equation adequativity is {}, with probability = {:.2f}\n".format(fishercheck,
0.95))

    console_output()
    if fishercheck:
        break
```

Результати роботи програми

```
Start. Current m = 3

Dispersion uniform is True, with probability = 0.95
Equation adequativity is True, with probability = 0.95
```

№	X1	X2	X3	X1*X2	X1*X3	X2*X3	X1*X2*X3	X1^2	X2^2	X3^2	Yi1	Yi2	Yi3	Ys	Ye
0	-5.000	-5.000	-4.000	25.000	20.000	20.000	-100.000	25.000	25.000	16.000	197	204	196	199.00	200.54
1	-5.000	-5.000	8.000	25.000	-40.000	-40.000	200.000	25.000	25.000	64.000	202	196	203	200.33	201.20
2	-5.000	6.000	-4.000	-30.000	20.000	-24.000	120.000	25.000	36.000	16.000	199	203	196	199.33	200.41
3	-5.000	6.000	8.000	-30.000	-40.000	48.000	-240.000	25.000	36.000	64.000	205	195	196	198.67	201.06
4	5.000	-5.000	-4.000	-25.000	-20.000	20.000	100.000	25.000	25.000	16.000	202	205	206	204.33	200.54
5	5.000	-5.000	8.000	-25.000	40.000	-40.000	-200.000	25.000	25.000	64.000	205	206	205	205.33	201.20
6	5.000	6.000	-4.000	30.000	-20.000	-24.000	-120.000	25.000	36.000	16.000	202	204	199	201.67	200.41
7	5.000	6.000	8.000	30.000	40.000	48.000	240.000	25.000	36.000	64.000	200	195	205	200.00	201.06
8	-6.075	0.500	2.000	-3.038	-12.150	1.000	-6.075	36.906	0.250	4.000	196	206	198	200.00	201.25
9	6.075	0.500	2.000	3.038	12.150	1.000	6.075	36.906	0.250	4.000	205	197	199	200.33	201.25
10	0.000	-7.183	2.000	-0.000	0.000	-14.365	-0.000	0.000	51.588	4.000	198	197	198	197.67	198.87
11	0.000	7.183	2.000	0.000	0.000	14.365	0.000	0.000	51.588	4.000	204	197	195	198.67	198.87
12	0.000	0.500	-9.290	0.000	-0.000	-4.645	-0.000	0.000	0.250	86.304	198	201	201	200.00	200.63
13	0.000	0.500	9.290	0.000	0.000	4.645	0.000	0.000	0.250	86.304	205	200	196	200.33	200.63
14	0.000	0.500	2.000	0.000	0.000	1.000	0.000	0.000	0.250	4.000	195	206	201	200.67	199.51

```
        Natural linear regrecy equation:  Y = 199.45584 + 0.04724*X1^2 + -0.01245*X2^2 + 0.01369*X3^2
```