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

Методи оптимізації та планування

Лабораторна робота №6

«Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами»

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, math
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, f):
    # take koefficient from start data array
    mat_y = [[round(sum([f[k] * combination_mul(xnat[i])[k] for k in range(11)]) +
random.randint(0, 10) - 5, 2)
              for j in range(m)]for i in range(n)]
    # counting Y middle
    for elem in mat_y:
        elem.append(sum(elem) / len(elem))
    return mat_y


# additional cycle for counting B koefficient
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,
           round(arr[0]*arr[1], 3),
           round(arr[0]*arr[2], 3),
           round(arr[1]*arr[2], 3),
           round(arr[0]*arr[1]*arr[2], 3),
           round(arr[0]*arr[0], 3),
           round(arr[1]*arr[1], 3),
           round(arr[2]*arr[2], 3)]


#calculate b in natural equation
def calcb(lmaty):
    # array from all combinations
    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]]]

    # generate system of linear equations
    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)
    # solving this
    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):
    # counting dispersion
    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
    # counting dispersion
    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():
    titles_x = ["№", "X1", "X2", "X3", "X1*X2", "X1*X3", "X2*X3", "X1*X2*X3", "X1^2", "X2^2",
"X3^2"]

    # normal=================================================

    # cycles for table with normal
    # title, combinations of Xnorm
    for j in range(11):
        s = ""
        if j == 0:
            s = "| {:^2s} |"
        if j >= 1 and j < 4:
            s = "{:^8s}|"
        if j >= 4 and j < 7:
            s = "{:^10s}|"
        if j == 7:
            s = "{:^11s}|"
        if j > 7 and j < 11:
            s = "{:^10s}|"
        print(s.format(titles_x[j]), end="")

    print()
    # aggregate for table, combinationns of Xnorm
    for i in range(n):
        print("| {:2d} |".format(i), end="")
        for j in range(1, 11):
            x = combination_mul(xnorm[i])[j]
            s = ""
            if j >= 1 and j < 4:
                s = "{:^ 8}|"
            if j >= 4 and j < 7:
                s = "{:^ 10}|"
            if j == 7:
                s = "{:^ 11}|"
            if j > 7 and j < 11:
                s = "{:^ 10}|"
            # using construction similar to ternar operator for printing 0, instead of 0.0
            print(s.format(x if x != 0 else 0), end="")
        print()
    print("\n")

    #natural=================================================================

    # cycles for table with natural
    # title, combinations of Xnat
    for j in range(11):
        s = ""
        if j == 0:
            s = "| {:^2s} |"
        if j >= 1 and j < 4:
            s = "{:^8s}|"
        if j >= 4 and j < 7:
            s = "{:^10s}|"
        if j == 7:
            s = "{:^11s}|"
        if j > 7 and j < 11:
            s = "{:^10s}|"
        print(s.format(titles_x[j]), end="")
```

```
    # title, Yi
    for i in range(m):
        print("{:^11s}|".format("Yi"+str(i+1)), end="")

    # title, middle Y and experimental Y
    print("{:^11s}|{:^11s}|".format("Ys", "Ye"), end="")

    print()
    # aggregate for table, combinationns of Xnat
    for i in range(n):
        print("| {:2d} |".format(i), end="")
        for j in range(1, 11):
            s = ""
            if j >= 1 and j < 4:
                s = "{:^ 8}|"
            if j >= 4 and j < 7:
                s = "{:^ 10}|"
            if j == 7:
                s = "{:^ 11}|"
            if j > 7 and j < 11:
                s = "{:^ 10}|"
            print(s.format(combination_mul(xnat[i])[j]), end="")

        # aggregate, Yi
        for j in maty[i][:-1]:
            print("{:^ 11}|".format(j), end="")
        # aggregate, middle Y, experimental Y
        print("{:^ 11}|{:^ 11}|"
              .format(maty[i][-1],
                      round(sum([combination_mul(xnat[i])[j] * b0[j] * d_arr[j] for j in
range(11)]), 2)), end="")

        print()

    print("\nNatural linear regrecy equation:\n\tY = ", end="")
    if d_arr[0] != 0:
        print("{:}".format(round(b0[0], 3)), end="")
    for i in range(1, 11):
        if d_arr[i] != 0:
            print(" {:+} * {}".format(round(b0[i], 3), titles_x[i]), end="")
    print()


n = 15
m = 2
l = 1.73

x1min = 15
x1max = 45
x01 = (x1min + x1max) / 2
xl1 = l*(x1max-x01)+x01

x2min = 15
x2max = 50
x02 = (x2min + x2max) / 2
xl2 = l*(x2max-x02)+x02

x3min = 15
x3max = 30
x03 = (x3min + x3max) / 2
xl3 = l*(x3max-x03)+x03

#            X1,  x2,  x3,  x1x2,  x1x3, x2x3, x1x2x3, x1^2, x2^2, x3^2
fxxx = [3.5, 6.6, 3.9, 1.8,   6,   0.8,   9.4,    3,   5.3,   0.5,   4.3]

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],
         [-l, 0, 0],
         [l, 0, 0],
         [0, -l, 0],
         [0, l, 0],
         [0, 0, -l],
         [0, 0, l],
         [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]]

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, fxxx)
        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()
    print("\nCount of meaningful koefficient, d = {}".format(d))
    if fishercheck:
        break
```

Результати роботи програми

Start. Current m = 2

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
0	-1	-1	-1	1	1	1	-1	1	1	1
1	-1	1	1	-1	-1	1	-1	1	1	1
2	1	-1	1	-1	1	-1	-1	1	1	1
3	1	1	-1	1	-1	-1	-1	1	1	1
4	-1	-1	1	1	-1	-1	1	1	1	1
5	-1	1	-1	-1	1	-1	1	1	1	1
6	1	-1	-1	-1	-1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	-1.73	0	0	0	0	0	0	2.993	0	0
9	1.73	0	0	0	0	0	0	2.993	0	0
10	0	-1.73	0	0	0	0	0	0	2.993	0
11	0	1.73	0	0	0	0	0	0	2.993	0
12	0	0	-1.73	0	0	0	0	0	0	2.993
13	0	0	1.73	0	0	0	0	0	0	2.993
14	0	0	0	0	0	0	0	0	0	0

№	X1	X2	X3	X1*X2	X1*X3	X2*X3	X1*X2*X3	X1^2	X2^2	X3^2	Yi1	Yi2	Ys	Ye
0	15	15	15	225	225	225	3375	225	225	225	16227.5	16227.5	16227.5	16227.26
1	15	15	30	225	450	450	6750	225	225	900	31575.0	31585.0	31580.0	31579.68
2	15	50	15	750	225	750	11250	225	2500	225	49216.5	49214.5	49215.5	49214.95
3	15	50	30	750	450	1500	22500	225	2500	900	93123.0	93121.0	93122.0	93121.37
4	45	15	15	675	675	225	10125	2025	225	225	49276.5	49279.5	49278.0	49278.63
5	45	15	30	675	1350	450	20250	2025	225	900	85240.0	85240.0	85240.0	85240.55
6	45	50	15	2250	675	750	33750	2025	2500	225	135816.5	135816.5	135816.5	135816.82
7	45	50	30	2250	1350	1500	67500	2025	2500	900	247585.0	247577.0	247581.0	247581.24
8	-55.95	32.5	22.5	-1818.375	-1258.875	731.25	-40913.438	3130.403	1056.25	506.25	-108690.3	-108687.3	-108688.8	-108688.66
9	55.95	32.5	22.5	1818.375	1258.875	731.25	40913.438	3130.403	1056.25	506.25	161371.57	161372.57	161372.07	161370.53
10	30.0	-62.775	22.5	-1883.25	675.0	-1412.438	-42373.125	900.0	3940.701	506.25	-142237.39	-142243.39	-142240.39	-142240.45
11	30.0	62.775	22.5	1883.25	675.0	1412.438	42373.125	900.0	3940.701	506.25	161644.84	161634.84	161639.84	161640.39
12	30.0	32.5	-35.475	975.0	-1064.25	-1152.938	-34588.125	900.0	1056.25	1258.476	-98629.43	-98625.43	-98627.43	-98627.44
13	30.0	32.5	35.475	975.0	1064.25	1152.938	34588.125	900.0	1056.25	1258.476	132406.07	132402.07	132404.07	132404.22
14	30.0	32.5	22.5	975.0	675.0	731.25	21937.5	900.0	1056.25	506.25	86919.0	86919.0	86919.0	86919.78

Natural linear regrecy equation:

Y = -7.584 +6.648 * X1 +4.129 * X2 +2.168 * X3 +6.002 * X1*X2 +0.8 * X1*X3 +9.39 * X2*X3 +3.0 * X1*X2*X3 +5.301 * X1^2 +0.5 * X2^2 +4.3 * X3^2

Count of meaningful koefficient, d = 11