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

Методи оптимізації та планування експерименту

Лабораторна робота № 6

Виконав:

студент групи ІО-91
Дикун А.В.

Залікова книжка № 9110

Варінт № 8
Перевірив: Регіда П. Г.

Київ 2021р

Текст програми:

```
import math
import numpy
import random
from _decimal import Decimal
from functools import reduce
from itertools import compress
from scipy.stats import f, t

# Ініціалізація змінних
xMin = [-30, -35, 0]
xMax = [0, 10, 20]
x0 = [(xMax[_] + xMin[_]) / 2 for _ in range(3)]
dx = [xMax[_] - x0[_] for _ in range(3)]
norm_plan_raw = [[-1, -1, -1],
                 [-1, +1, +1],
                 [+1, -1, +1],
                 [+1, +1, -1],
                 [-1, -1, +1],
                 [-1, +1, -1],
                 [+1, -1, -1],
                 [+1, +1, +1],
                 [-1.73, 0, 0],
                 [+1.73, 0, 0],
                 [0, -1.73, 0],
                 [0, +1.73, 0],
                 [0, 0, -1.73],
                 [0, 0, +1.73]]

naturalPlanRaw = [[xMin[0], xMin[1], xMin[2]],
                  [xMin[0], xMin[1], xMax[2]],
                  [xMin[0], xMax[1], xMin[2]],
                  [xMin[0], xMax[1], xMax[2]],
                  [xMax[0], xMin[1], xMin[2]],
                  [xMax[0], xMin[1], xMax[2]],
                  [xMax[0], xMax[1], xMin[2]],
                  [xMax[0], xMax[1], xMax[2]],
                  [-1.73 * dx[0] + x0[0], x0[1], x0[2]],
                  [1.73 * dx[0] + x0[0], x0[1], x0[2]],
                  [x0[0], -1.73 * dx[1] + x0[1], x0[2]],
                  [x0[0], 1.73 * dx[1] + x0[1], x0[2]],
                  [x0[0], x0[1], -1.73 * dx[2] + x0[2]],
                  [x0[0], x0[1], 1.73 * dx[2] + x0[2]],
                  [x0[0], x0[1], x0[2]]]


# Основні функції
def equationRegres(x1, x2, x3, coefficients, importance=[True] * 11):
    factors_array = [1, x1, x2, x3, x1 * x1, x2 * x2, x3 * x3, x1 * x2, x1 * x3, x2 *
x3, x1 * x2 * x3]
    return sum([el[0] * el[1] for el in compress(zip(coefficients, factors_array),
importance)])


def func(x1, x2, x3):
    coefficients = [5.4, 3.6, 6.6, 7.7, 8, 0.3, 2.5, 5.9, 0.3, 7.2, 5.3]
    return equationRegres(x1, x2, x3, coefficients)


def generateFactorsTable(rawArr):
    raw_list = [row + [row[0] * row[1], row[0] * row[2], row[1] * row[2], row[0] *
row[1] * row[2]] + list(
        map(lambda x: x ** 2, row)) for row in rawArr]
    return list(map(lambda row: list(map(lambda el: round(el, 3), row)), raw_list))
```

```
def generateY(m, factorsTable):
    return [[round(func(row[0], row[1], row[2]) + random.randint(-5, 5), 3) for _ in
range(m)] for row in factorsTable]


# Вивід результатів
def printMatrix(m, n, factors, valsY, additionalText=":"):
    labels_table = list(map(lambda x: x.ljust(10),
                            ["x1", "x2", "x3", "x12", "x13", "x23", "x123", "x1^2",
"x2^2", "x3^2"] + [
                                "y{}".format(i + 1) for i in range(m)]))
    rows_table = [list(factors[i]) + list(valsY[i]) for i in range(n)]
    print("Матриця планування" + additionalText)
    print(" ".join(labels_table))
    print("\n".join([" ".join(map(lambda j: "{:<+10}".format(j), rows_table[i])) for i
in range(len(rows_table))]))
    print("\t")


def printEquation(coefficients, importance=[True] * 11):
    XiNames = list(compress(["", "x1", "x2", "x3", "x12", "x13", "x23", "x123", "x1^2",
"x2^2", "x3^2"], importance))
    coefficientsToPrint = list(compress(coefficients, importance))
    equation = " ".join(
        ["".join(i) for i in zip(list(map(lambda x: "{:+.2f}".format(x),
coefficientsToPrint)), XiNames)])
    print("Рівняння регресії: y = " + equation)


def setFactorsTable(factorsTable):
    def x_i(i):
        withNullFactor = list(map(lambda x: [1] + x,
generateFactorsTable(factorsTable)))
        res = [row[i] for row in withNullFactor]
        return numpy.array(res)

    return x_i


def Mij(*arrays):
    return numpy.average(reduce(lambda accum, el: accum * el, list(map(lambda el:
numpy.array(el), arrays))))


def findCoeffs(factors, y_values):
    Xi = setFactorsTable(factors)
    coefficients = [[Mij(Xi(column), Xi(row)) for column in range(11)] for row in
range(11)]
    numpyY = list(map(lambda row: numpy.average(row), y_values))
    freeVal = [Mij(numpyY, Xi(i)) for i in range(11)]
    betaCoeffs = numpy.linalg.solve(coefficients, freeVal)
    return list(betaCoeffs)


# Критерії
def cochranCriteria(m, n, y_table):
    def getCochranVal(f1, f2, q):
        part_result1 = q / f2
        params = [part_result1, f1, (f2 - 1) * f1]
        fisher = f.isf(*params)
        result = fisher / (fisher + (f2 - 1))
        return Decimal(result).quantize(Decimal('.0001')).__float__()

    print("Перевірка рівномірності дисперсій за критерієм Кохрена: m = {}, N =
{}".format(m, n))
    variationsY = [numpy.var(i) for i in y_table]
    variationMaxY = max(variationsY)
```

```
    gp = variationMaxY / sum(variationsY)
    f1 = m - 1
    f2 = n
    p = 0.95
    q = 1 - p
    gt = getCochranVal(f1, f2, q)
    print("Gp = {}; Gt = {}; f1 = {}; f2 = {}; q = {:.2f}".format(gp, gt, f1, f2, q))
    if gp < gt:
        print("Gp < Gt => дисперсії рівномірні - все правильно")
        return True
    else:
        print("Gp > Gt => дисперсії нерівномірні - треба ще експериментів")
        return False


def studentCriteria(m, n, y_table, betaCoeffs):
    def getStudentVal(f3, q):
        return Decimal(abs(t.ppf(q / 2, f3))).quantize(Decimal('.0001')).__float__()

    print("Перевірка значимості коефіцієнтів регресії за критерієм Стьюдента: m = {}, N
= {} ".format(m, n))
    averageVariation = numpy.average(list(map(numpy.var, y_table)))
    variationBetaS = averageVariation / n / m
    standardDeviationBetaS = math.sqrt(variationBetaS)
    Ti = [abs(betaCoeffs[i]) / standardDeviationBetaS for i in range(len(betaCoeffs))]
    f3 = (m - 1) * n
    q = 0.05
    ourT = getStudentVal(f3, q)
    importance = [True if el > ourT else False for el in list(Ti)]
    # print result data
    print("Оцінки коефіцієнтів βs: " + ", ".join(list(map(lambda x: str(round(float(x),
3)), betaCoeffs))))
    print("Коефіцієнти ts: " + ", ".join(list(map(lambda i: "{:.2f}".format(i), Ti))))
    print("f3 = {}; q = {}; tтабл = {}".format(f3, q, ourT))
    betaI = ["β0", "β1", "β2", "β3", "β12", "β13", "β23", "β123", "β11", "β22", "β33"]
    importanceToPrint = ["важливий" if i else "неважливий" for i in importance]
    toPrint = map(lambda x: x[0] + " " + x[1], zip(betaI, importanceToPrint))
    print(*toPrint, sep="; ")
    printEquation(betaCoeffs, importance)
    return importance


def fisherCriteria(m, N, d, tableX, tableY, coeffsB, importance):
    def getFisherVal(f3, f4, q):
        return Decimal(abs(f.isf(q, f4, f3))).quantize(Decimal('.0001')).__float__()

    f3 = (m - 1) * N
    f4 = N - d
    q = 0.05
    theorY = numpy.array([equationRegres(row[0], row[1], row[2], coeffsB) for row in
tableX])
    avgY = numpy.array(list(map(lambda el: numpy.average(el), tableY)))
    Sad = m / (N - d) * sum((theorY - avgY) ** 2)
    variationsY = numpy.array(list(map(numpy.var, tableY)))
    Sv = numpy.average(variationsY)
    Fp = float(Sad / Sv)
    Ft = getFisherVal(f3, f4, q)
    theoreticalValsToPrint = list(
        zip(map(lambda x: "x1 = {0[1]:<10} x2 = {0[2]:<10} x3 = {0[3]:<10}".format(x),
tableX), theorY))
    print("Перевірка адекватності моделі за критерієм Фішера: m = {}, N = {} для
таблиці y_table".format(m, N))
    print("Теоретичні значення y для різних комбінацій факторів:")
    print("\n".join(["{arr[0]}: y = {arr[1]}".format(arr=el) for el in
theoreticalValsToPrint]))
    print("Fp = {}, Ft = {}".format(Fp, Ft))
    print("Fp < Ft => модель адекватна" if Fp < Ft else "Fp > Ft => модель
```

```
неадекватна")
    return True if Fp < Ft else False


def main(m, n):
    naturalPlan = generateFactorsTable(naturalPlanRaw)
    arrY = generateY(m, naturalPlanRaw)
    while not cochranCriteria(m, n, arrY):
        m += 1
        arrY = generateY(m, naturalPlan)

    printMatrix(m, n, naturalPlan, arrY, " для натуралізованих факторів:")
    coefficients = findCoeffs(naturalPlan, arrY)
    printEquation(coefficients)
    importance = studentCriteria(m, n, arrY, coefficients)
    d = len(list(filter(None, importance)))
    fisherCriteria(m, n, d, naturalPlan, arrY, coefficients, importance)


if __name__ == "__main__":
    m = 3
    n = 15
    main(m, n)
```

Результати роботи програми:

```
C:\Users\anvat\Anaconda3\envs\MOPE_labs\python.exe C:/Users/anvat/PycharmProjects/MOPE_labs/Lab6/main.py
Перевірка рівномірності дисперсій за критерієм Кохрена: m = 3, N = 15
Gp = 0.1772428884026258; Gt = 0.3346; f1 = 2; f2 = 15; q = 0.05
Gp < Gt => дисперсії рівномірні - все правильно
Матриця планування для натуралізованих факторів:
x1        x2        x3        x12       x13       x23       x123      x1^2      x2^2      x3^2      y1          y2          y3
-30       -35       +0        +1050     +0        +0        +0        +900      +1225     +0        +13427.9    +13426.9    +13431.9
-30       -35       +20       +1050     -600      -700      +21000    +900      +1225     +400      +120662.9   +120664.9   +120667.9
-30       +10       +0        -300      +0        +0        +0        +900      +100      +0        +5422.4     +5427.4     +5418.4
-30       +10       +20       -300      -600      +200      -6000     +900      +100      +400      -23961.6    -23965.6    -23961.6
+0        -35       +0        +0        +0        +0        +0        +0        +1225     +0        +146.9      +146.9      +137.9
+0        -35       +20       +0        +0        -700      +0        +0        +1225     +400      -3749.1     -3748.1     -3746.1
+0        +10       +0        +0        +0        +0        +0        +0        +100      +0        +100.4      +102.4      +105.4
+0        +10       +20       +0        +0        +200      +0        +0        +100      +400      +2696.4     +2694.4     +2695.4
-40.95    -12.5     +10.0     +511.875  -409.5    -125.0    +5118.75  +1676.903 +156.25   +100.0    +42690.163  +42689.163  +42688.163
+10.95    -12.5     +10.0     -136.875  +109.5    -125.0    -1368.75  +119.902  +156.25   +100.0    -7630.672   -7635.672   -7629.672
-15.0     -51.425   +10.0     +771.375  -150.0    -514.25   +7713.75  +225.0    +2644.531 +100.0    +44221.742  +44223.742  +44214.742
-15.0     +26.425   +10.0     -396.375  -150.0    +264.25   -3963.75  +225.0    +698.281  +100.0    -19024.598  -19031.598  -19023.598
-15.0     -12.5     -7.3      +187.5    +109.5    +91.25    -1368.75  +225.0    +156.25   +53.29    -3663.485   -3669.485   -3670.485
-15.0     -12.5     +27.3     +187.5    -409.5    -341.25   +5118.75  +225.0    +156.25   +745.29   +29439.985  +29444.985  +29439.985
-15.0     -12.5     +10.0     +187.5    -150.0    -125.0    +1875.0   +225.0    +156.25   +100.0    +12137.525  +12137.525  +12139.525
```

```
Рівняння регресії: y = +5.81 +3.87x1 +6.65x2 +7.53x3 +5.90x12 +0.29x13 +7.20x23 +5.30x123 +8.01x1^2 +0.30x2^2 +2.50x3^2
Перевірка значимості коефіцієнтів регресії за критерієм Стьюдента: m = 3, N = 15
Оцінки коефіцієнтів βs: 5.808, 3.867, 6.653, 7.528, 5.9, 0.295, 7.205, 5.3, 8.005, 0.303, 2.503
Коефіцієнти ts: 14.97, 9.97, 17.15, 19.41, 15.21, 0.76, 18.57, 13.66, 20.64, 0.78, 6.45
f3 = 30; q = 0.05; tтабл = 2.0423
β0 важливий; β1 важливий; β2 важливий; β3 важливий; β12 важливий; β13 неважливий; β23 важливий; β123 важливий; β11 важливий; β22 неважливий; β33 важливий
Рівняння регресії: y = +5.81 +3.87x1 +6.65x2 +7.53x3 +5.90x12 +7.20x23 +5.30x123 +8.01x1^2 +2.50x3^2
Перевірка адекватності моделі за критерієм Фішера: m = 3, N = 15 для таблиці y_table
Теоретичні значення y для різних комбінацій факторів:
x1 = -35        x2 = 0          x3 = 1050      : y = 10893.401074138506
x1 = -35        x2 = 20         x3 = 1050      : y = 61482.036384222025
x1 = 10         x2 = 0          x3 = -300      : y = 3705.7432008480705
x1 = 10         x2 = 20         x3 = -300      : y = -13024.89695626667
x1 = -35        x2 = 0          x3 = 0         : y = 134.1692274394426
x1 = -35        x2 = 20         x3 = 0         : y = 2954.5647182601347
x1 = 10         x2 = 0          x3 = 0         : y = 101.82802081566822
x1 = 10         x2 = 20         x3 = 0         : y = 3194.9081225134987
x1 = -12.5      x2 = 10.0       x3 = 511.875   : y = 22711.09468652377
x1 = -12.5      x2 = 10.0       x3 = -136.875  : y = -1799.0322014936883
x1 = -51.425    x2 = 10.0       x3 = 771.375   : y = 24551.21522038198
x1 = 26.425     x2 = 10.0       x3 = -396.375  : y = -10691.734529102701
x1 = -12.5      x2 = -7.3       x3 = 187.5     : y = 38.7049697291809
x1 = -12.5      x2 = 27.3       x3 = 187.5     : y = 17239.86606715244
x1 = -12.5      x2 = 10.0       x3 = 187.5     : y = 6482.956492514936
Fp = 351599927.93939227, Ft = 2.4205
Fp > Ft => модель неадекватна

Process finished with exit code 0
```