МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний Технічний Університет України
«Київський Політехнічний Інститут»
Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №2
з дисципліни «Методи оптимізації та планування»
на тему: «ПРОВЕДЕННЯ ДВОФАКТОРНОГО ЕКСПЕРИМЕНТУ З ВИКОРИСТАННЯМ ЛІНІЙНОГО РІВНЯННЯ РЕГРЕСІЇ»

Виконав:
студент 2-го курсу ФІОТ
групи ІВ-71
Мазан Я. В.

Перевірив:
Асистент
Регіда П. Г.

Київ – 2019

Варіант:

№ у списку - 9

№ варіанту	x_1		x_2	
	min	max	min	max
109	-20	15	10	60

$y_{max} = 210$

$y_{min} = 110$

Код програми:

```
import itertools
import numpy as np
from random import *
import math
from functools import *
"""
constants
"""
y_max = (30 - 9)*10           # 210
y_min = (20 - 9)*10           # 110
x_table = [[-1,-1],
           [-1,+1],
           [+1,-1]]
p = 0.99
x1_min = -20
x1_max = 15
x2_min = 10
x2_max = 60
naturalized_x_table = [[x1_min, x2_min],
                       [x1_min, x2_max],
                       [x1_max,  x2_min]]
"""
Romanovsky criteria start
"""
def romanovsky_criteria(y1: np.array, y2: np.array, y3: np.array):
    def sigma_theta(m):
        return math.sqrt(abs(2*(2*m-2)/(m*(m-4))))
    def f_uv(y_u: np.array, y_v: np.array):
        dev_u = np.var(y_u)
        dev_v = np.var(y_v)
        return dev_u/dev_v if dev_u > dev_v else dev_v/dev_u
    def theta_uv(m: int, fuv: float):
        return (m-2)/m * fuv
    def r_uv(s_t: float, s_uv: float):
        return abs(s_uv - 1)/s_t
    def check_criteria(R, m):
        romanovsky_criteria_table = [[None, 2,    6,    8,    10,   12,   15,   20],
                                     [0.99, 1.72, 2.16, 2.43, 2.62, 2.75, 2.90,
3.08],
                                     [0.98, 1.72, 2.13, 2.37, 2.54, 2.66, 2.80,
2.96],
                                     [0.95, 1.71, 2.10, 2.27, 2.41, 2.52, 2.64,
2.78],
                                     [0.90, 1.69, 2.00, 2.17, 2.29, 2.39, 2.49,
2.62]]
        column = romanovsky_criteria_table[0].index(sorted(filter(lambda el: el >= m,
romanovsky_criteria_table[0][1:]))[0])
        # in our case equals 1 (p = 0.99)
        trusted_probability_row = 1
        # defines necessary row depending of given p. Everything works,
        # but it's unnecessary to add so much functionality in this labwork
```

```
        # global p
        # trusted_probabilities = [row[0] for row in romanovsky_criteria_table[1:]]
        # trusted_probability_row = trusted_probabilities.index(min(filter(lambda el:
el >= p, trusted_probabilities)))
        # trusted_probability_row = 1 if trusted_probability_row == 0 else
trusted_probability_row
        return R < romanovsky_criteria_table[trusted_probability_row][column]
    global m
    sTheta = sigma_theta(m)
    accordance = True
    for combination in itertools.combinations((y1,y2,y3), 2):
        fUV = f_uv(combination[0], combination[1])
        sUV = theta_uv(m, fUV)
        R = r_uv(sTheta,sUV)
        accordance *= check_criteria(R,m)
    return accordance
# Romanovsky criteria was tested for these experiment values (all rows' variations are
drastically different)
# to ensure correctness of our romanovsky_criteria() function:
#
# m = 5
# y_table = [[-5,10000,1,1,1], [3,4,5,1,2], [4, -16,-20,3,8]]
# all deviations of rows in this y_table are drastically different, so
romanovsky_criteria(y_table) must be false
# result of test == false
# everything works correctly
"""
Romanovsky criteria end
Regression coefficients search start
"""
def experiment():
    global m
    return np.array([[randint(y_min, y_max) for _ in range(m)] for _ in range(3)])
def normalized_regression_coeffs():
    def m_i(arr: np.array):
        return np.average(arr)
    def a_i(arr: np.array):
        return sum(arr**2)/len(arr)
    def a_jj(arr1: np.array, arr2: np.array):
        return reduce(lambda res, el: res+el[0]*el[1], list(zip(arr1,arr2)),
0)/len(arr1)
    global x_table
    global y_table
    y_vals = np.array([np.average(i) for i in y_table])
    x1_vals = np.array([i[0] for i in x_table])
    x2_vals = np.array([i[1] for i in x_table])
    m_x1 = m_i(x1_vals)
    m_x2 = m_i(x2_vals)
    m_y  = m_i(y_vals)
    a1   = a_i(x1_vals)
    a2   = a_jj(x1_vals, x2_vals)
    a3   = a_i(x2_vals)
    a11  = a_jj(x1_vals, y_vals)
    a22  = a_jj(x2_vals, y_vals)
    coeffs_matrix = [[1, m_x1, m_x2],
                     [m_x1, a1, a2],
                     [m_x2, a2, a3]]
    vals_matrix = [m_y, a11, a22]
    b_coeffs = list(map(lambda num: round(num, 2), np.linalg.solve(coeffs_matrix,
vals_matrix)))
    return  b_coeffs
def assert_normalized_regression():
    global b_coeffs
    global x_table
    global y_table
```

```
    y_average_experim_vals = np.array([np.average(i) for i in y_table])
    print("\nПеревірка правильності знаходження коефіцієнтів рівняння регресії: ")
    print("Середні експериментальні значення у для кожного рядка матриці планування: " +
          ", ".join(map(str, y_average_experim_vals)))
    y_theoretical = [b_coeffs[0] + x_table[i][0]*b_coeffs[1] + x_table[i][1]*b_coeffs[2]
for i in range(len(x_table))]
    print("Теоретичні значення у для кожного рядка матриці планування: ".ljust(74) + ",
".join(map(str, y_theoretical)))
    for i in range(len(x_table)):
        try:
            assert round(y_theoretical[i], 2) == round(y_average_experim_vals[i],2)
        except:
            print("Неправильні результати пошуку коефіцієнтів рівняння регресії")
            return
    print("Правильні результати пошуку коефіцієнтів рівняння регресії")
"""
Regression coefficients search end
"""
def naturalized_regression(b_coeffs: list):
    v = globals()
    global x1_max
    global x1_min
    global x2_max
    global x2_min
    x1 = abs(x1_max-x1_min)/2
    x2 = abs(x2_max-x2_min)/2
    x10 = (x1_max+x1_min)/2
    x20 = (x2_max+x2_min)/2
    a0 = b_coeffs[0]-b_coeffs[1]*x10/x1 - b_coeffs[2]*x20/x2
    a1 = b_coeffs[1]/x1
    a2 = b_coeffs[2]/x2
    return [a0, a1, a2]
def assert_naturalized_regression():
    global y_table
    global naturalized_x_table
    global a_coeffs
    y_average_experim_vals = np.array([np.average(i) for i in y_table])
    print("\nПеревірка натуралізації коефіцієнтів рівняння регресії:")
    print("Середні експериментальні значення у для кожного рядка матриці планування: " +
          ", ".join(map(str, y_average_experim_vals)))
    y_theoretical = [a_coeffs[0] + naturalized_x_table[i][0]*a_coeffs[1]+
naturalized_x_table[i][1]*a_coeffs[2] for i in range(len(naturalized_x_table))]
    print("Теоретичні значення у для кожного рядка матриці планування: ".ljust(74) + ",
".join(
        map(str, y_theoretical)))
    for i in range(len(naturalized_x_table)):
        try:
            assert round(y_theoretical[i],2) == round(y_average_experim_vals[i],2)
        except:
            print("Неправильні результати натуралізації")
            return
    print("Правильні результати натуралізації")
m = 5
y_table = experiment()
while not romanovsky_criteria(*y_table):
    m += 1
    y_table = experiment()
labels_table = ["x1", "x2"] + ["y{}".format(i+1) for i in range(m)]
rows_table = [naturalized_x_table[i] + list(y_table[i]) for i in range(3)]
rows_normalized_table = [x_table[i] + list(y_table[i]) for i in range(3)]
print("Матриця планування:")
print((" "*4).join(labels_table))
print("\n".join([" ".join(map(lambda j: "{:<+5}".format(j), rows_table[i])) for i in
range(len(rows_table))]))
print("\t")
```

```
print("Нормована матриця планування:")
print((" "*4).join(labels_table))
print("\n".join([" ".join(map(lambda j: "{:<+5}".format(j), rows_normalized_table[i]))
for i in range(len(rows_normalized_table))]))
print("\t")
b_coeffs = normalized_regression_coeffs()
print("Рівняння регресії для нормованих факторів: y = {0} {1:+}*x1
{2:+}*x2".format(*b_coeffs))
assert_normalized_regression()
a_coeffs = naturalized_regression(b_coeffs)
print("\nРівняння регресії для натуралізованих факторів: y = {0} {1:+}*x1
{2:+}*x2".format(*a_coeffs))
assert_naturalized_regression()
```

Результати виконання:

```
/home/yan/PycharmProjects/Optimization&PlanningLab2/venv/bin/python /home/yan/PycharmProjects/Optimization&PlanningLab2/m
Матриця планування:
x1    x2    y1    y2    y3    y4    y5
-20   +10   +200  +114  +188  +186  +124
-20   +60   +138  +185  +189  +146  +150
+15   +10   +165  +131  +111  +150  +196

Нормована матриця планування:
x1    x2    y1    y2    y3    y4    y5
-1    -1    +200  +114  +188  +186  +124
-1    +1    +138  +185  +189  +146  +150
+1    -1    +165  +131  +111  +150  +196

Рівняння регресії для нормованих факторів: y = 156.1 -5.9*x1 -0.4*x2

Перевірка правильності знаходження коефіцієнтів рівняння регресії:
Середні експериментальні значення y для кожного рядка матриці планування: 162.4, 161.6, 150.6
Теоретичні значення y для кожного рядка матриці планування:               162.4, 161.6, 150.6
Правильні результати пошуку коефіцієнтів рівняння регресії

Рівняння регресії для натуралізованих факторів: y = 155.81714285714284 -0.3371428571428572*x1 -0.016*x2

Перевірка натуралізації коефіцієнтів рівняння регресії:
Середні експериментальні значення y для кожного рядка матриці планування: 162.4, 161.6, 150.6
Теоретичні значення y для кожного рядка матриці планування:               162.39999999999998, 161.59999999999997, 150.6
Правильні результати натуралізації

Process finished with exit code 0
```