Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

ЛАБОРАТОРНА РОБОТА № 6

з дисципліни «Методи наукових досліджень»

на тему «Проведення трьохфакторного експерименту

при використанні рівняння регресії квадратичними членами.»

ВИКОНАВ:

студент 2 курсу

групи ІВ-91  
Желепа Валентин

Залікова – 9110

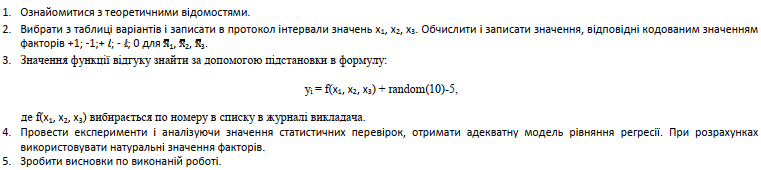
ПЕРЕВІРИВ:

ас. Регіда П. Г.

Київ – 2021

**Мета:** Провести повний трьохфакторний експеримент і отримати адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план.

**Завдання на лабораторну роботу**

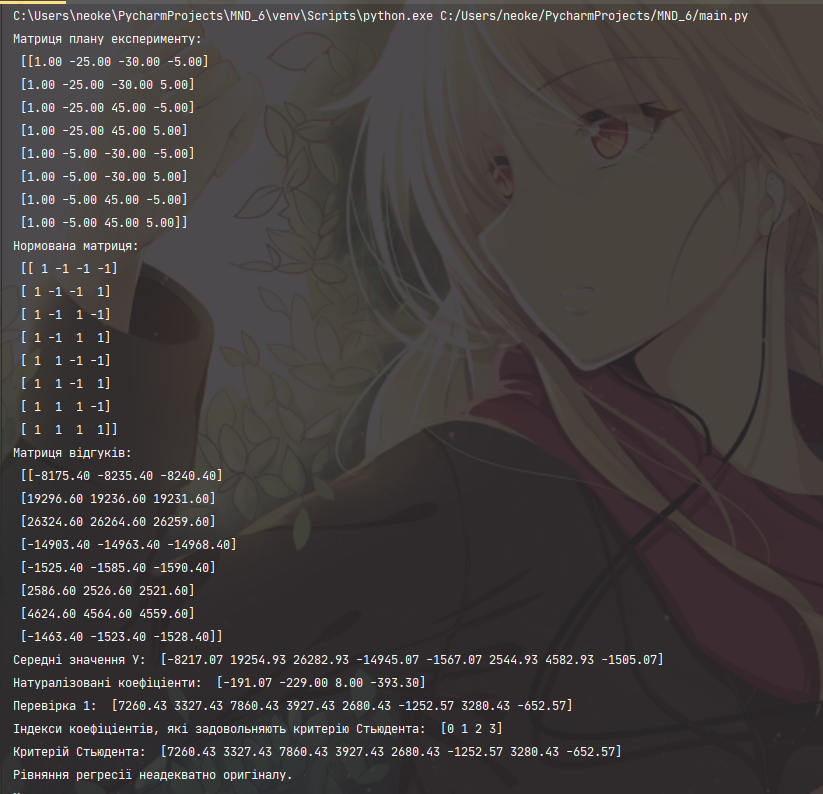


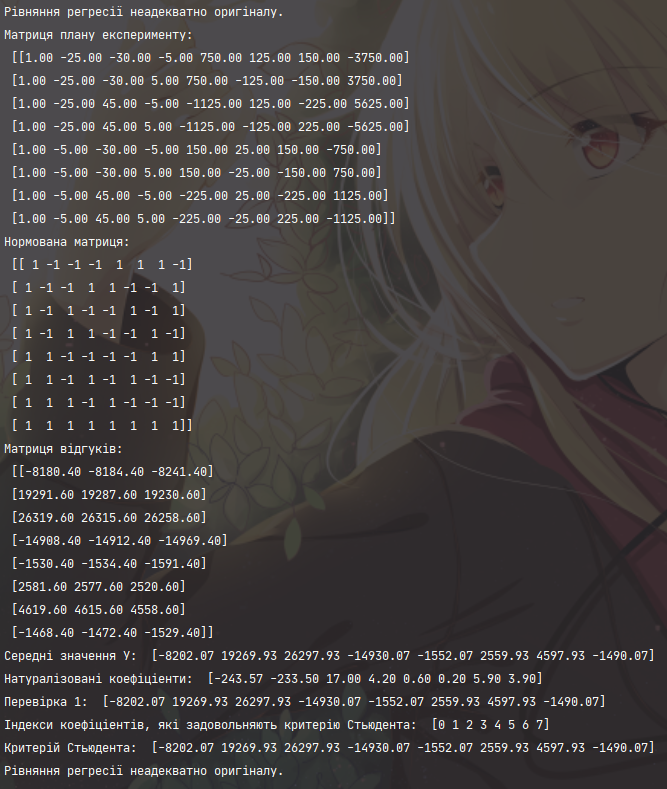
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № варіанту | x1 | | x2 | | x3 | |
| min | max | min | max | min | max |
| 110 | -25 | -5 | -30 | 45 | -5 | 5 |

**Програмний код**

*import* random  
*import* numpy *as* np  
*from* itertools *import* product, combinations  
  
np.set\_printoptions(*formatter*={'float\_kind': *lambda* x: "%.2f" % (x)})  
  
Tt = 1.45  
Ft = 2.16  
Gt = 0.3346  
gt = {12: {1: 0.5410, 2: 0.3924, 3: 0.3264, 4: 0.2880, 5: 0.2624, 6: 0.2439, 7: 0.2299, 8: 0.2187, 9: 0.2098, 10: 0.2020},  
 15: {1: 0.4709, 2: 0.3346, 3: 0.2758, 4: 0.2419, 5: 0.2159, 6: 0.2034, 7: 0.1911, 8: 0.1815, 9: 0.1736, 10: 0.1671}}  
tt = {24: 2.064, 30: 2.042, 32: 1.96} *# m = [3, 6]*ft = {1: 4.2, 2: 3.3, 3: 2.9, 4: 2.7, 5: 2.5, 6: 2.4}  
minXmax = np.array([[-25, -5], [-30, 45], [-5, 5]])  
m = 3  
  
*def* studentsTtest(*normMatrix*, *matrixY*, *N*):  
 meanY = np.mean(*matrixY*, *axis*=1)  
 dispersion = np.mean((*matrixY*.T - meanY) \*\* 2, *axis*=0)  
 meanDispers = np.mean(dispersion)  
 sigma = np.sqrt(meanDispers / (*N* \* m))  
 betta = np.mean(*normMatrix*.T \* meanY, *axis*=1)  
 t = np.abs(betta) / sigma  
 *return* np.where(t > Tt)  
  
  
*def* phisherCriterion(*yMatrix*, *d*, *N*):  
 *if d* == *N*:  
 *return False* Sad = (m / (*N* - *d*)) \* np.sum((check2 - yMean)\*\*2)  
 meanDispers = np.mean(np.mean((*yMatrix*.T - yMean) \*\* 2, *axis*=0))  
 Fp = Sad / meanDispers  
 *return* Fp < Ft  
  
*def* cochranCheck(*matrixY*, N):  
 meanY = np.mean(*matrixY*, *axis*=1)  
 dispersion = np.mean((*matrixY*.T - meanY) \*\* 2, *axis*=0)  
 Gp = np.max(dispersion) / (np.sum(dispersion))  
 *return* Gp < Gt  
  
  
*def* makePlanMatrixFromNormMatrix(*normMatrix*):  
 planMatrix = np.empty((*len*(*normMatrix*), *len*(*normMatrix*[0])), *dtype*=np.float)  
 *for* i *in range*(*len*(*normMatrix*)):  
 *for* j *in range*(*len*(*normMatrix*[i])):  
 *if normMatrix*[i, j] == -1:  
 planMatrix[i, j] = minXmax[j-1][0]  
 *elif normMatrix*[i, j] == 1 *and* j != 0:  
 planMatrix[i, j] = minXmax[j-1][1]  
 *elif normMatrix*[i, j] == 1 *and* j == 0:  
 planMatrix[i, j] = 1  
 *else*:  
 mean = np.mean(minXmax[j-1])  
 planMatrix[i, j] = *normMatrix*[i, j] \* (minXmax[j-1][1] - mean) + mean  
 *return* planMatrix  
  
  
*def* makeLinearEquation():  
 normMatrix = np.array(*list*(product("01", *repeat*=3)), *dtype*=np.int)  
 normMatrix[normMatrix == 0] = -1  
 normMatrix = np.insert(normMatrix, 0, 1, *axis*=1)  
 planMatrix = makePlanMatrixFromNormMatrix(normMatrix)  
 *return* normMatrix, planMatrix  
  
  
*def* makeEquationWithInteractionEffect(*currentNormMatrix*, *currentPlanMatrix*):  
 planMatr = *currentPlanMatrix* normMatrix = *currentNormMatrix* combination = *list*(combinations(*range*(1, 4), 2))  
 *for* i *in* combination:  
 planMatr = np.append(planMatr, np.reshape(planMatr[:, i[0]] \* planMatr[:, i[1]], (*len*(normMatrix), 1)),*axis*=1)  
 normMatrix = np.append(normMatrix, np.reshape(normMatrix[:, i[0]] \* normMatrix[:, i[1]], (*len*(normMatrix), 1)), *axis*=1)  
 planMatr = np.append(planMatr, np.reshape(planMatr[:, 1] \* planMatr[:, 2] \* planMatr[:, 3], (*len*(normMatrix), 1)), *axis*=1)  
 normMatrix = np.append(normMatrix, np.reshape(normMatrix[:, 1] \* normMatrix[:, 2] \* normMatrix[:, 3], (*len*(normMatrix), 1)), *axis*=1)  
 *return* normMatrix, planMatr  
  
  
*def* makeEquationWithQuadraticTerms(*currentNormMatrix*):  
 normMatrixSecondPart = np.empty((3, 7))  
 key = 0  
 *for* i *in range*(3):  
 j = 0  
 *while* j < 7:  
 *if* j == key:  
 normMatrixSecondPart[i][key] = -1.73  
 normMatrixSecondPart[i][key + 1] = 1.73  
 j += 1  
 *else*:  
 normMatrixSecondPart[i][j] = 0  
 j += 1  
 key += 2  
  
 normMatrixSecondPart = np.insert(normMatrixSecondPart, 0, 1, *axis*=0)  
 normMatrix = np.append(*currentNormMatrix*, normMatrixSecondPart.T, *axis*=0)  
 planMatrix = makePlanMatrixFromNormMatrix(normMatrix)  
 planMatrix = makeEquationWithInteractionEffect(normMatrix, planMatrix)[1]  
 planMatrix = np.append(planMatrix, planMatrix[:, 1:4] \*\* 2, *axis*=1)  
 normMatrix = makeEquationWithInteractionEffect(normMatrix, planMatrix)[0]  
 normMatrix = np.append(normMatrix, normMatrix[:, 1:4] \*\* 2, *axis*=1)  
 *return* normMatrix, planMatrix  
  
  
count = 0  
flagOfModel = *False  
while* flagOfModel *is False*:  
 normMatrix = makeLinearEquation()[0]  
 planMatr = makeLinearEquation()[1]  
 *if* count == 1:  
 normMatrix = makeEquationWithInteractionEffect(normMatrix, planMatr)[0]  
 planMatr = makeEquationWithInteractionEffect(normMatrix, planMatr)[1]  
 *elif* count > 1:  
 planMatr = makeEquationWithQuadraticTerms(normMatrix)[1]  
 normMatrix = makeEquationWithQuadraticTerms(normMatrix)[0]  
 planMatrForCalcY = planMatr  
 N = *len*(planMatr)  
 yMatrix = []  
 yMean = []  
 indexes = []  
 flagOfDispersion = *False  
 while* flagOfDispersion *is False*:  
 yMatrix = np.array(  
 [8.6 + 6.5 \* planMatrForCalcY[:, 1] + 9.5 \* planMatrForCalcY[:, 2] + 4.2 \* planMatrForCalcY[:,3] + 8.0 \* planMatrForCalcY[:, 1] \*\* 2 +  
 0.5 \* planMatrForCalcY[:, 2] \*\* 2 + 2.3 \* planMatrForCalcY[:, 3] \*\* 2 + 0.6 \* planMatrForCalcY[:,1] \* planMatrForCalcY[:, 2] +  
 0.2 \* planMatrForCalcY[:, 1] \* planMatrForCalcY[:, 3] + 5.9 \* planMatrForCalcY[:,2] \* planMatrForCalcY[:, 3] +  
 3.9 \* planMatrForCalcY[:, 1] \* planMatrForCalcY[:, 2] \* planMatrForCalcY[:, 3] + random.randint(0, 100) - 50 *for* i *in range*(m)]).T  
 yMean = np.mean(yMatrix, *axis*=1)  
 *if* cochranCheck(yMatrix, N):  
 flagOfDispersion = *True* bNatura = np.linalg.lstsq(planMatr, yMean, *rcond*=*None*)[0]  
 bNorm = np.linalg.lstsq(normMatrix, yMean, *rcond*=*None*)[0]  
 check1 = np.sum(bNatura \* planMatr, *axis*=1)  
 indexes = studentsTtest(normMatrix, yMatrix, N)  
 check2 = np.sum(bNatura[indexes] \* np.reshape(planMatr[:, indexes], (N, np.size(indexes))), *axis*=1)  
 *print*("Матриця плану експерименту: \n", planMatr)  
 *print*("Нормована матриця: \n", normMatrix)  
 *print*("Матриця відгуків: \n", yMatrix)  
 *print*("Середні значення У: ", yMean)  
 *print*("Натуралізовані коефіціенти: ", bNatura)  
 *print*("Перевірка 1: ", check1)  
 *print*("Індекси коефіціентів, які задовольняють критерію Стьюдента: ", np.array(indexes)[0])  
 *print*("Критерій Стьюдента: ",check2)  
 *else*:  
 m += 1  
 *print*("Дисперсія неоднорідна!")  
 *if* phisherCriterion(yMatrix, np.size(indexes), N):  
 flagOfModel = *True  
 print*("Рівняння регресії адекватно оригіналу.")  
 *else*:  
 count += 1  
 *print*("Рівняння регресії неадекватно оригіналу.")

**Результат роботи програми**

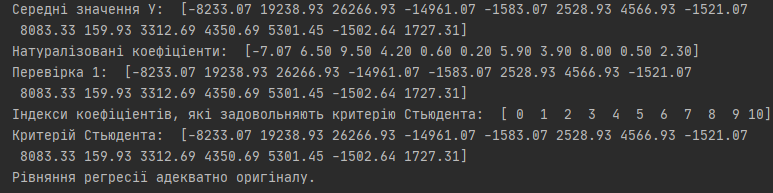
****

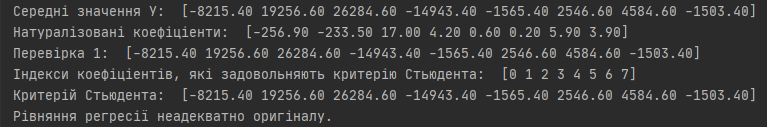
****

****

****

Позитивний результат



Негативний результат 

**Висновок:**

Під час виконання даної лабораторної роботи я провів трьохфакторний експеримент під час якого потрібно отримати адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план, перевірив однорідність дисперсії за критерієм Кохрена, отримав коефіцієнти рівняння регресії, оцінив значимість знайдених коефіцієнтів за критеріями Стьюдента та Фішера.

Мета лабораторної роботи була досягнута.