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:

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

Виконав:

студент групи ІО-83

Соловйов Даніїл Олександрович

Залікова книжка № 8324

Перевірив Регіда П. Г.

Київ 2020р.

**Лабораторна робота №6**

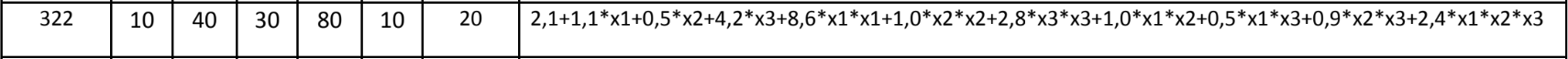
**Тема:** Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами

**Мета:** провести трьохфакторний експеримент і отримати адекватну модель – рівняння регресії, використовуючи рототабельний

композиційний план.

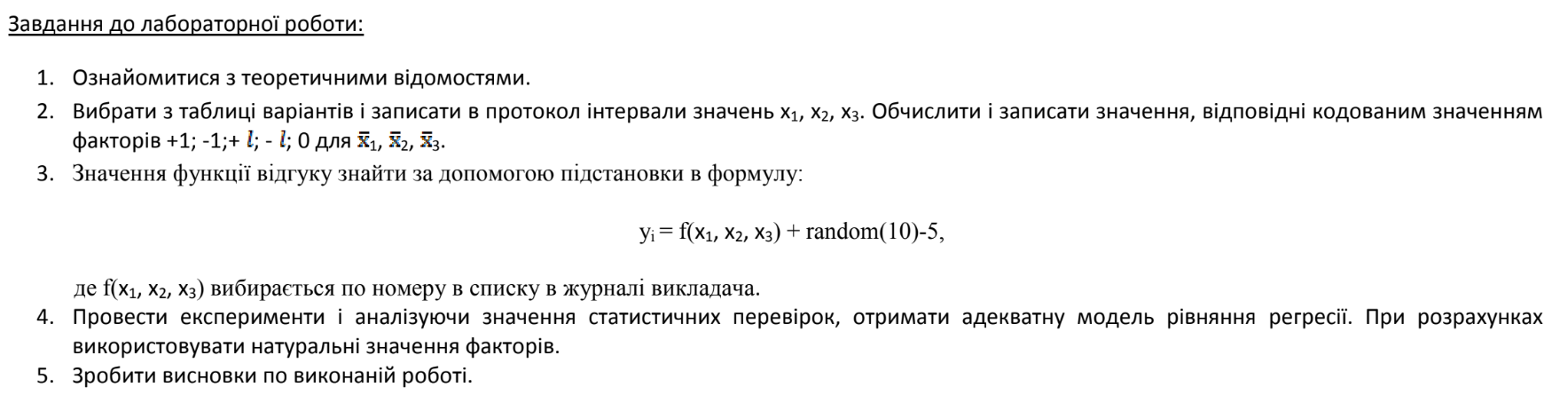
**Виконання:**

Варіант – 322.

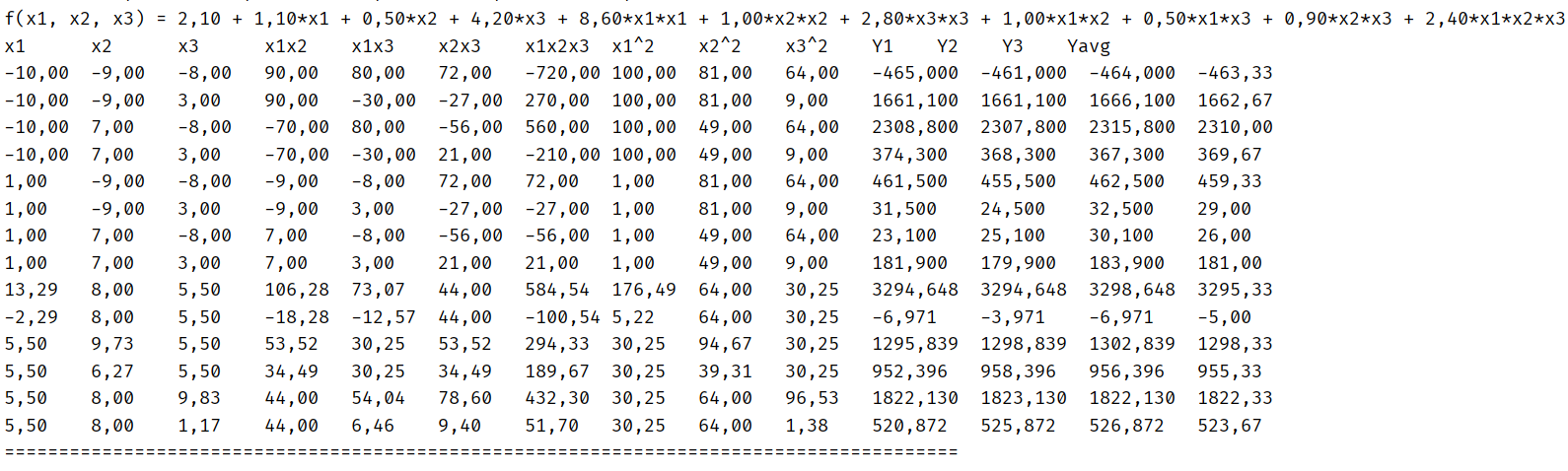


x1min = 10; x2min = 30; x3min = 10;

x1max = 40; x2min = 80; x3min = 20;



Матриця планування з ефектом взаємодії та урахуванням квадратичних членів:



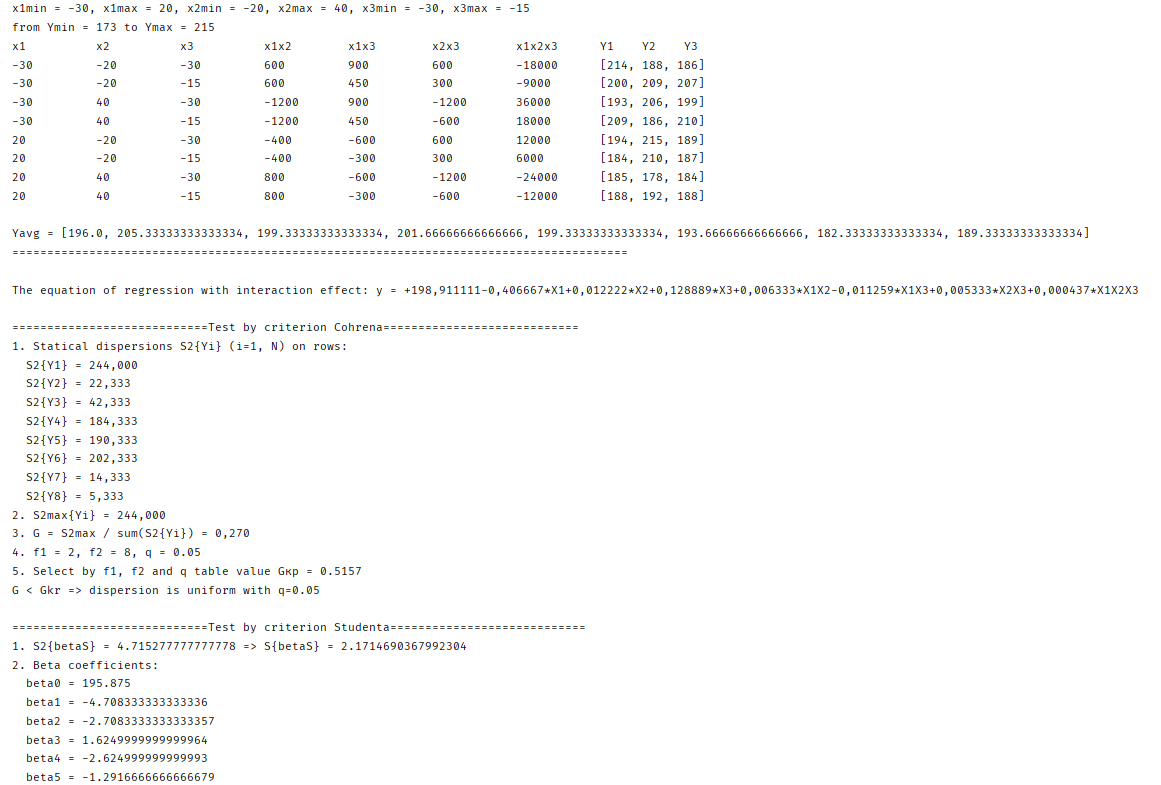
*// Спочатку запускаємо програму із Лабораторної роботи №4 для того, щоб перевірити адекватність моделі  
// з ефектом взаємодії. Якщо модель не адекватна оригіналу, то запускаємо вже програму безпоседеньо із  
// Лабораторної роботи №6 ModuleWithSq.class для запуску програми із ефектом взаємодії та квадратними членами  
// використовуючи рототабельний комп. план.*

1. Лістинг програми:

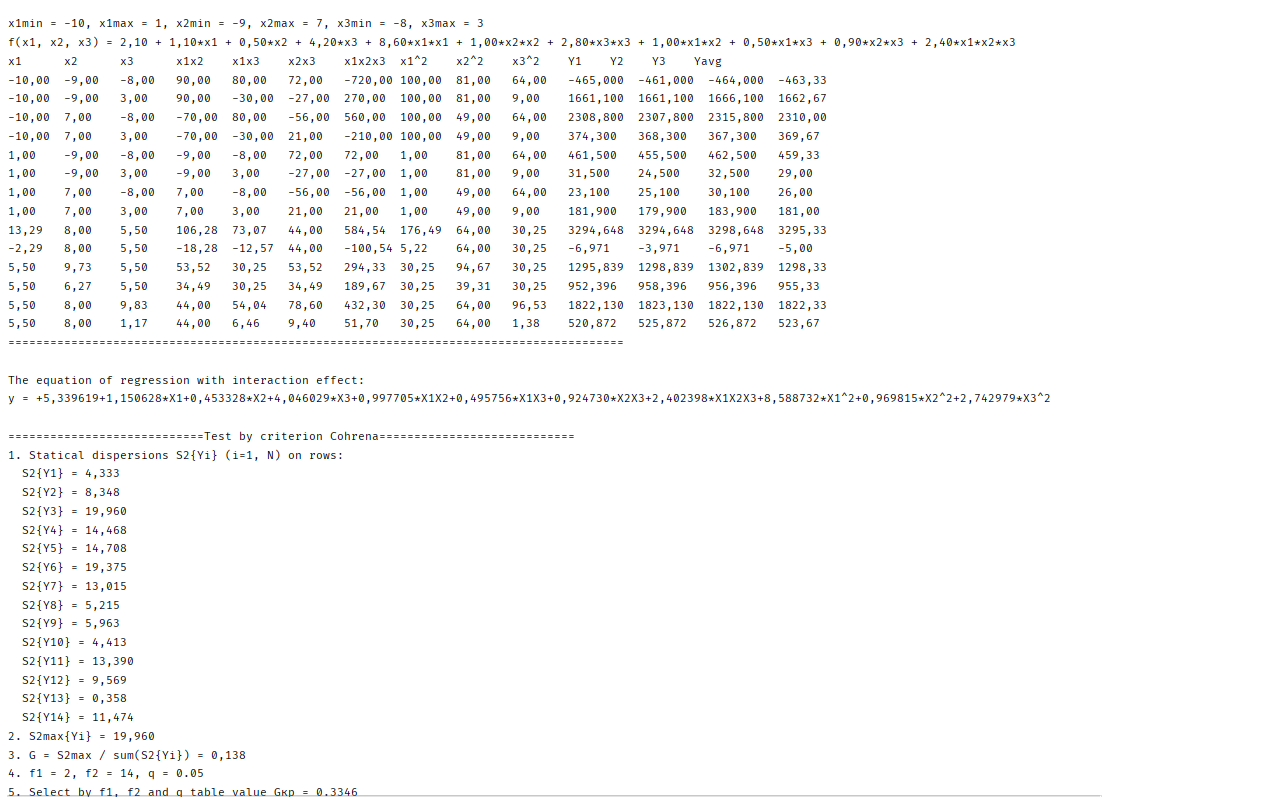
**import** java.util.Arrays;  
  
**public class** ModuleWithSq {  
 **static int** getAvg(**int**[] a) {  
 **return** (**int**)Math.*round*(Arrays.*stream*(a).average().getAsDouble());  
 }  
  
 **public static void** main(String[] args) {  
 **int**[] xMinMax = {-10, 1, -9, 7, -8, 3};  
 **double**[] fCoef = {2.1, 1.1, 0.5, 4.2, 8.6, 1.0, 2.8, 1.0, 0.5, 0.9, 2.4};  
  
 System.***out***.println(**"x1min = "** + xMinMax[0] + **", x1max = "** + xMinMax[1] + **", x2min = "** + xMinMax[2] +  
 **", x2max = "** + xMinMax[3] + **", x3min = "** + xMinMax[4] + **", x3max = "** + xMinMax[5]);  
  
 System.***out***.printf(**"f(x1, x2, x3) = %.2f + %.2f\*x1 + %.2f\*x2 + %.2f\*x3 + %.2f\*x1\*x1 + %.2f\*x2\*x2 + %.2f\*x3\*x3 "** +  
 **"+ %.2f\*x1\*x2 + %.2f\*x1\*x3 + %.2f\*x2\*x3 + %.2f\*x1\*x2\*x3\n"**,fCoef[0], fCoef[1], fCoef[2], fCoef[3], fCoef[4],  
 fCoef[5], fCoef[6], fCoef[7], fCoef[8], fCoef[9], fCoef[10]);  
  
  
 ThreeFactorsExperimentWithSq exp = **new** ThreeFactorsExperimentWithSq(xMinMax, fCoef);  
 exp.printMatrixOfPlanning();  
 exp.findEquationOfRegression();  
 exp.testByCriterionKohrena();  
 exp.testByStudentCriterion();  
 exp.testByFisheraCriterion();  
 }  
}

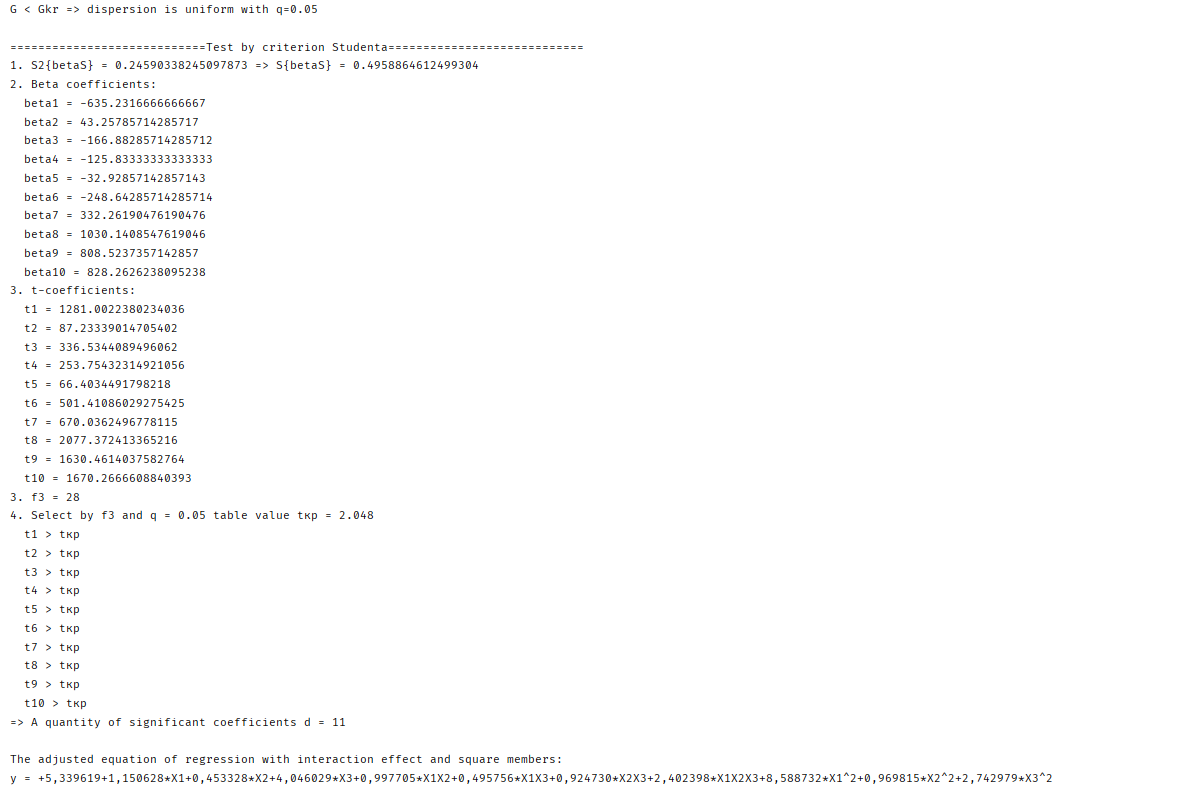
**import** org.apache.commons.math3.linear.\*;  
  
**import** java.util.Arrays;  
**import** java.util.Map;  
**import** java.util.Random;  
  
**public class** ThreeFactorsExperimentWithSq {  
 **private int N** = 14;  
 **private static int** *m* = 3;  
 **private double**[][] **xk** = { *// Кодовані значення x* {-1, -1, -1, -1, 1, 1, 1, 1, -1.73, 1.73, 0, 0, 0, 0}, *// xk1* {-1, -1, 1, 1, -1, -1, 1, 1, 0, 0, -1.73, 1.73, 0, 0}, *// xk2* {-1, 1, -1, 1, -1, 1, -1, 1, 0, 0, 0, 0, -1.73, 1.73}, *// xk3* {1, 1, -1, -1, -1, -1, 1, 1, 0, 0, 0, 0, 0, 0}, *// xk12* {1, -1, 1, -1, -1, 1, -1, 1, 0, 0, 0, 0, 0, 0}, *// xk13* {1, -1, -1, 1, 1, -1, -1, 1, 0, 0, 0, 0, 0, 0}, *// xk23* {-1, 1, 1, -1, 1, -1, -1, 1, 0, 0, 0, 0, 0, 0}, *// xk123* {1, 1, 1, 1, 1, 1, 1, 1, 2.9929, 2.9929, 0, 0, 0, 0}, *// xk1^2* {1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 2.9929, 2.9929, 0, 0}, *// xk1^2* {1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 2.9929, 2.9929}, *// xk1^2* };  
 **private double**[][] **y**;  
 **private double**[] **f**;  
 **private double**[][] **x**;  
 **private int**[] **X**;  
 **private double**[] **Yavg** = **new double**[**N**]; *// Середні значення Yi по рядках, i=1..N* **private double**[] **b** = **new double**[11]; *// Значення b коефіцієнтів* **private double**[] **S2y**, **y\_**; *// Коефіцієнти для перевірок* **private int d**; *// Кількість значущих коеф.* **private double** f(**double** x1, **double** x2, **double** x3) {  
 **return f**[0]+**f**[1]\*x1+**f**[2]\*x2+**f**[3]\*x3+**f**[4]\*x1\*x1+**f**[5]\*x2\*x2+**f**[6]\*x3\*x3+**f**[7]\*x1\*x2+**f**[8]\*x1\*x3+**f**[9]\*x2\*x3  
 +**f**[10]\*x1\*x2\*x3;  
 }  
  
 *// Метод для відображення матриці планування ПФЕ* **public void** printMatrixOfPlanning() {  
 System.***out***.printf(**"%-8s%-8s%-8s%-8s%-8s%-8s%-8s%-8s%-8s%-8s"**,  
 **"x1"**, **"x2"**,**"x3"**,**"x1x2"**,**"x1x3"**,**"x2x3"**,**"x1x2x3"**, **"x1^2"**, **"x2^2"**, **"x3^2"**);  
 **for** (**int** i = 0; i < *m*; i++) {  
 System.***out***.printf(**"Y%-5d"**, i + 1);  
 }  
 System.***out***.print(**"Yavg"**);  
 System.***out***.println();  
 **for** (**int** i = 0; i < **N**; i++) {  
 **for** (**int** j = 0; j < **x**.**length**; j++) {  
 System.***out***.printf(**"%-5.2f\t"**, **x**[j][i]);  
 }  
 **for** (**int** j = 0; j < *m*; j++) {  
 System.***out***.printf(**"%-10.3f"**,**y**[i][j]);  
 }  
 System.***out***.printf(**"%5.2f\n"**, **Yavg**[i]);  
 }  
 System.***out***.println(**"========================================================================================"**);  
 }  
  
 *// Конструктор класу* **public** ThreeFactorsExperimentWithSq(**int**[] X, **double**[] f) {  
 **if** (X.**length** != 6) {  
 **throw new** RuntimeException(**"The length of array 'x' must be equaled 6! But founded "** + X.**length**);  
 }  
 **this**.**f** = f;  
 **this**.**X** = X;  
 generateMatrixOfPlanning(*m*);  
 }  
  
 *// Метод для створення матрицы планування* **private void** generateMatrixOfPlanning(**int** m) {  
 Random random = **new** Random();  
 **d** = 1;  
 **y** = **new double**[**N**][m];  
 **x** = **new double**[10][**N**];  
 **int** total = 0;  
  
  
 *// Заповнення матриці від x1 до x3* **for** (**int** i = 0, k = 0, v = 0; i < 3; i++, k += 2, v+=2) {  
 **for** (**int** j = 0; j < **N**; j++) {  
 **if** (j < 8)  
 **x**[i][j] = (**xk**[i][j] == -1) ? **X**[k] : **X**[k + 1];  
 **else** {  
 **double** ximax = **X**[v+1], ximin = **X**[v];  
 **double** dx = ximax - (ximax-ximin)/2.0;  
 **double** x0i = (ximax-ximin)/2.0;  
 **x**[i][j] = **xk**[i][j]\*dx+x0i;  
 }  
 }  
 }  
  
 *// Заповнення матриці від x1x2 до x1x2x3* **int** t = 3;  
 **for** (**int** i = 0; i < 3; i++) {  
 **for** (**int** j = i+1; j < 3; j++) {  
 **for** (**int** k = 0; k < **N**; k++) {  
 **x**[t][k] = **x**[i][k]\***x**[j][k];  
 }  
 t++;  
 }  
 }  
 **for** (**int** i = 0; i < **N**; i++) {  
 **x**[6][i] = **x**[0][i]\***x**[1][i]\***x**[2][i];  
 }  
  
 *// Заповнення матриці від x1^2 до x3^2* **for** (**int** i = 7, v = 0; i < 10; i++, v++) {  
 **for** (**int** j = 0; j < **N**; j++) {  
 **x**[i][j] = **x**[v][j]\***x**[v][j];  
 }  
 }  
  
  
 **for** (**int** i = 0; i < **y**.**length**; i++) {  
 **for** (**int** j = 0; j < **y**[i].**length**; j++) {  
 **y**[i][j] = f(**x**[0][i], **x**[1][i], **x**[2][i])+random.nextInt(10) - 5;  
 total += **y**[i][j];  
 }  
 **Yavg**[i] = (**double**) total / m;  
 total = 0;  
 }  
 }  
  
 *// Проміжний метод для знаходження сум по стовпцях матриці планування* **private double** s(**int** v1, **int** v2) {  
 **double** sum = 0;  
 **for** (**int** i = 0; i < **N**; i++) {  
 sum+=**x**[v1-1][i]\***x**[v2-1][i];  
 }  
 **return** sum;  
 }  
  
 *// Метод для знаходження коефіцієнтів рівняння регресії* **public void** findEquationOfRegression() {  
 **double**[] k = **new double**[11];  
 **double**[] mx = **new double**[10];  
  
 **for** (**int** i = 0; i < mx.**length**; i++) {  
 mx[i] = Arrays.*stream*(**x**[i]).sum();  
 }  
  
 **double**[][] m = {  
 {**N**, mx[0], mx[1], mx[2], mx[3], mx[4], mx[5], mx[6], mx[7], mx[8], mx[9] },  
 {mx[0], s(1, 1), s(1, 2), s(1, 3), s(1, 4), s(1, 5), s(1, 6), s(1, 7), s(1, 8), s(1, 9), s(1, 10)},  
 {mx[1], s(2, 1), s(2, 2), s(2, 3), s(2, 4), s(2, 5), s(2, 6), s(2, 7), s(2, 8), s(2, 9), s(2, 10)},  
 {mx[2], s(3, 1), s(3, 2), s(3, 3), s(3, 4), s(3, 5), s(3, 6), s(3, 7), s(3, 8), s(3, 9), s(3, 10)},  
 {mx[3], s(4, 1), s(4, 2), s(4, 3), s(4, 4), s(4, 5), s(4, 6), s(4, 7), s(4, 8), s(4, 9), s(4, 10)},  
 {mx[4], s(5, 1), s(5, 2), s(5, 3), s(5, 4), s(5, 5), s(5, 6), s(5, 7), s(5, 8), s(5, 9), s(5, 10)},  
 {mx[5], s(6, 1), s(6, 2), s(6, 3), s(6, 4), s(6, 5), s(6, 6), s(6, 7), s(6, 8), s(6, 9), s(6, 10)},  
 {mx[6], s(7, 1), s(7, 2), s(7, 3), s(7, 4), s(7, 5), s(7, 6), s(7, 7), s(7, 8), s(7, 9), s(7, 10)},  
 {mx[7], s(8, 1), s(8, 2), s(8, 3), s(8, 4), s(8, 5), s(8, 6), s(8, 7), s(8, 8), s(8, 9), s(8, 10)},  
 {mx[8], s(9, 1), s(9, 2), s(9, 3), s(9, 4), s(9, 5), s(9, 6), s(9, 7), s(9, 8), s(9, 9), s(9, 10)},  
 {mx[9], s(10, 1), s(10, 2), s(10, 3), s(10, 4), s(10, 5), s(10, 6), s(10, 7), s(10, 8), s(10, 9), s(10, 10)}  
 };  
  
 k[0] = Arrays.*stream*(**Yavg**).sum();  
 **for** (**int** i = 1; i < k.**length**; i++) {  
 **for** (**int** j = 0; j < **N**; j++) {  
 k[i] += **Yavg**[j] \* **x**[i - 1][j];  
 }  
 }  
  
 RealMatrix coefficients = **new** Array2DRowRealMatrix(m, **false**);  
 DecompositionSolver solver = **new** LUDecomposition(coefficients).getSolver();  
 RealVector constants = **new** ArrayRealVector(k, **false**);  
 RealVector solution = solver.solve(constants);  
  
 **for** (**int** i = 0; i < solution.toArray().**length**; i++) {  
 **b**[i] = solution.getEntry(i);  
 }  
  
 *// Рівняння регресії з ефектом взаємодії* System.***out***.printf(**"\nThe equation of regression with interaction effect:\ny = %+f%+f\*X1%+f\*X2%+f\*X3%+f\*X1X2%+f\*X1X3"** +  
 **"%+f\*X2X3%+f\*X1X2X3%+f\*X1^2%+f\*X2^2%+f\*X3^2\n"**, **b**[0], **b**[1], **b**[2], **b**[3], **b**[4], **b**[5], **b**[6], **b**[7],  
 **b**[8], **b**[9], **b**[10]);  
 }  
  
 *// Перша статистична перевірка* **public void** testByCriterionKohrena() {  
 **double** q = 0.05;  
  
 *// для рядка при f2 = 15 при q=0.005. Таблиця побудована із одним рядком, адже f2 буде завжди дорівнювати 15, а  
 // f1, у свою чергу, буде дорівнювати 3, а може бути і 4 і тд. Тож, не має сенсу будувати усю таблицю.* **double**[][] CohrenaTable = {  
 {.4709, .3346, .2758, .2419, .2159, .2034, .1911, .1815, .1736, .1671},  
 };  
  
 System.***out***.println(**"\n============================Test by criterion Cohrena============================"**);  
 System.***out***.println(**"1. Statical dispersions S2{Yi} (i=1, N) on rows: "**);  
  
 **S2y** = **new double**[**N**];  
 **for** (**int** i = 0; i < **y**.**length**; i++) {  
 **double** s = 0;  
 **for** (**int** j = 0; j < **y**[i].**length**; j++) {  
 s += (**y**[i][j] - **Yavg**[i])\*(**y**[i][j] - **Yavg**[i]);  
 }  
 **S2y**[i] = s/(*m*-1);  
 System.***out***.printf(**" S2{Y%d} = %.3f\n"**,i+1,**S2y**[i]);  
 }  
 **double** S2max = Arrays.*stream*(**S2y**).max().getAsDouble();  
  
 System.***out***.printf(**"2. S2max{Yi} = %.3f\n"**,S2max);  
  
 **double** G = S2max/Arrays.*stream*(**S2y**).sum();  
 System.***out***.printf(**"3. G = S2max / sum(S2{Yi}) = %.3f\n"**,G);  
  
 **int** f1 = *m* - 1, f2 = **N**;  
 System.***out***.println(**"4. f1 = "**+f1+**", f2 = "**+f2+**", q = "**+q);  
  
 **double** Gkr = CohrenaTable[0][f1 - 1];  
 System.***out***.println(**"5. Select by f1, f2 and q table value Gкр = "**+Gkr);  
  
 **if** (G < Gkr)  
 System.***out***.println(**"G < Gkr => dispersion is uniform with q="**+q);  
 **else** {  
 System.***out***.println(**"G >= Gkr => dispersion is not uniform with q="** + q+**". So, m = m + 1 = "**+(++*m*)+**"\n"**);  
 generateMatrixOfPlanning(*m*);  
 printMatrixOfPlanning();  
 findEquationOfRegression();  
 testByCriterionKohrena();  
 }  
 }  
  
 *// Друга статистична перевірка* **public void** testByStudentCriterion() {  
 System.***out***.println(**"\n============================Test by criterion Studenta============================"**);  
 Map<Integer, Double> StudentaTable = Map.*of*(26, 2.056, 27, 2.052,28, 2.048,29,  
 2.044,30,2.042, 45, 2.011, 60, 2.0,  
 75, 1.994); *// Таблиця для p=1-q=0.95 починаючи з f=30* **double** S2beta = Arrays.*stream*(**S2y**).sum()/(**N**\***N**\**m*);  
 **double**[] beta = **new double**[10];  
 **double**[] t = **new double**[10];  
 **double** q = 0.05;  
  
 System.***out***.println(**"1. S2{betaS} = "**+S2beta+**" => S{betaS} = "**+Math.*sqrt*(S2beta));  
 System.***out***.println(**"2. Beta coefficients: "**);  
 **for** (**int** i = 0; i < 10; i++) {  
 **for** (**int** j = 0; j < **xk**[i].**length**; j++) {  
 beta[i]+=(**Yavg**[j]\***xk**[i][j]);  
 }  
 beta[i] /= **N**;  
 t[i] = Math.*abs*(beta[i]) / Math.*sqrt*(S2beta);  
 System.***out***.println(**" beta"**+ (i+1) +**" = "**+beta[i]);  
 }  
  
 System.***out***.println(**"3. t-coefficients: "**);  
 **for** (**int** i = 0; i < t.**length**; i++) {  
 System.***out***.println(**" t"**+(i+1)+**" = "**+t[i]);  
 }  
  
 **int** f3 = (*m*-1)\***N**;  
 **double** tkr = StudentaTable.get(f3);  
 System.***out***.println(**"3. f3 = "**+f3);  
 System.***out***.println(**"4. Select by f3 and q = "**+q+**" table value tкр = "**+tkr);  
 **for** (**int** i = 0; i < t.**length**; i++) {  
 **if** (t[i] < tkr) {  
 System.***out***.printf(**" t%d < tкр\n"**, (i+1));  
 **b**[i+1] = 0;  
 }  
 **else** {  
 System.***out***.printf(**" t%d > tкр\n"**, (i+1));  
 **d**++;  
 }  
 }  
 System.***out***.println(**"=> A quantity of significant coefficients d = "**+**d**);  
  
 *// Скореговане рівняння регресії з ефектом взаємодії та квадратними членами* System.***out***.printf(**"\nThe adjusted equation of regression with interaction effect and square members:\n"** +  
 **"y = %+f%+f\*X1%+f\*X2%+f\*X3%+f\*X1X2%+f\*X1X3"** +  
 **"%+f\*X2X3%+f\*X1X2X3%+f\*X1^2%+f\*X2^2%+f\*X3^2\n"**, **b**[0], **b**[1], **b**[2], **b**[3], **b**[4], **b**[5], **b**[6],  
 **b**[7], **b**[8], **b**[9], **b**[10]);  
  
 **y\_** = **new double**[**N**];  
 System.***out***.println(**"\nSubstitute the coded values of 'x' into the regression equation:"**);  
 **for** (**int** i = 0; i < **N**; i++) {  
 **y\_**[i] = **b**[0]+**b**[1]\***x**[0][i]+**b**[2]\***x**[1][i]+**b**[3]\***x**[2][i]+**b**[4]\***x**[3][i]+**b**[5]\***x**[4][i]+**b**[6]\***x**[5][i]+**b**[7]\***x**[6][i];  
 System.***out***.printf(**"Y\_%d = %.3f\n"**,i+1,**y\_**[i]);  
 }  
 }  
  
 *// Третя статистична перевірка* **public void** testByFisheraCriterion() {  
 System.***out***.println(**"\n============================Test by criterion Fishera============================"**);  
 *// Таблиця крітерію Фішера, починаючи з рядка для f3 = 30* **double**[][] FisheraTable = {  
 {4.2, 3.3, 2.9, 2.7, 2.5, 2.4, 2.35, 2.3, 2.25, 2.2, 2.15, 2.1}, *// для 28* {4.2, 3.3, 2.9, 2.7, 2.5, 2.4, 2.35, 2.3, 2.25, 2.2, 2.15, 2.1}, *// для 30* {4.1, 3.2, 2.9, 2.6, 2.5, 2.3, 2.25, 2.2, 2.15, 2.1, 2.0}, *// для 45* {4.0, 3.2, 2.8, 2.5, 2.4, 2.3, 2.25, 2.2, 2.15, 2.1, 2.0}, *// для 60* };  
 **int** f4 = **N** - **d**;  
 **int** f3 = (*m*-1)\***N**;  
 **double** q = 0.05;  
  
 **double** sum = 0;  
 **for** (**int** i = 0; i < **Yavg**.**length**; i++) {  
 sum+=(**Yavg**[i] - **y\_**[i])\*(**y\_**[i] - **Yavg**[i]);  
 }  
 **double** S2ad = (**double**) *m*/(**N**-**d**)\*sum;  
 **double** Fp = S2ad/(Arrays.*stream*(**S2y**).sum()/**N**); *// Fp = S²ад / S²в* System.***out***.println(**"3. f4 = N - d = "**+f4+**", f3 = "**+f3);  
  
 **double** Ft = 0;  
 **switch** (f3) {  
 **case** 28:  
 Ft = FisheraTable[0][f4-1];  
 **break**;  
 **case** 30:  
 Ft = FisheraTable[1][f4-1];  
 **break**;  
 **case** 45:  
 Ft = FisheraTable[2][f4-1];  
 **break**;  
 **case** 60:  
 Ft = FisheraTable[3][f4-1];  
 **break**;  
 }  
 System.***out***.println(**"4. Select by f3, f4 and q = "**+q+**" table value Ft = "**+Ft);  
 **if** (Fp <= Ft)  
 System.***out***.println(**"Fp <= Ft => So, the equation is adequate to original with q = "**+q);  
 **else** {  
 System.***out***.println(**"Fp > Ft => So, the equation is inadequate to original with q = "** + q+**". \n"** +  
 **"Since the equation is not adequate to the original, we start the analysis again.\n\n"**);  
 *m* = 3;  
 generateMatrixOfPlanning(*m*);  
 printMatrixOfPlanning();  
 findEquationOfRegression();  
 testByCriterionKohrena();  
 testByStudentCriterion();  
 testByFisheraCriterion();  
 }  
 }  
}

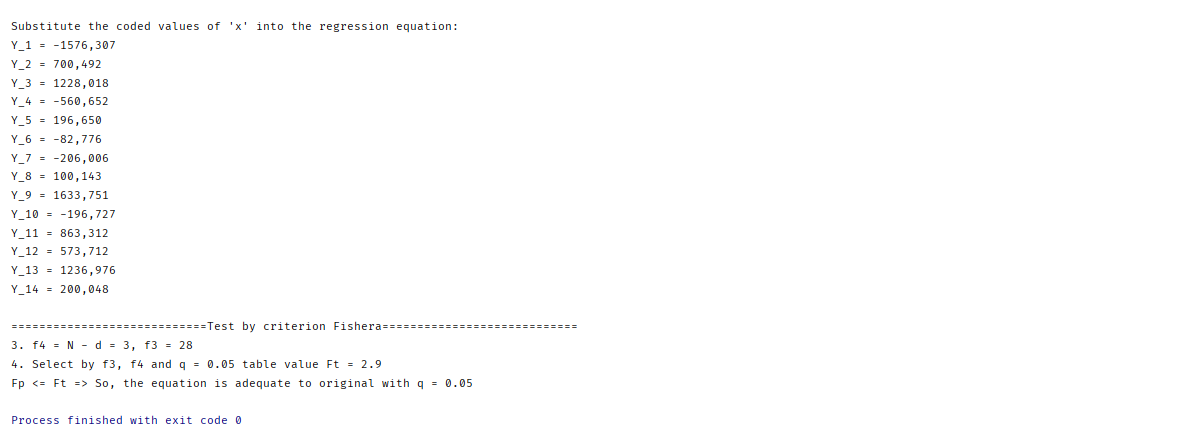
1. Результат виконання роботи програми:











**Висновок:** Отже, у ході виконання лабораторної роботи №6 провели трьохфакторний експеримент і отримали адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план. Була написана відповідна текстова програма. Результати наведені вище. Кінцева мета роботи досягнута!