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 №3:

«ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ З ВИКОРИСТАННЯМ

ЛІНІЙНОГО РІВНЯННЯ РЕГРЕСІЇ»

Виконав:

студент групи ІО-83

Соловйов Даніїл Олександрович

Залікова книжка № 8324

Перевірив Регіда П. Г.

Київ 2020р.

**Лабораторна робота №3**

**Тема:** ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ З ВИКОРИСТАННЯМ

ЛІНІЙНОГО РІВНЯННЯ РЕГРЕСІЇ

**Мета:** провести дробовий трьохфакторний експеримент. Скласти матрицю планування, знайти

коефіцієнти рівняння регресії, провести 3 статистичні перевірки.

**Виконання:**

Варіант – 322.

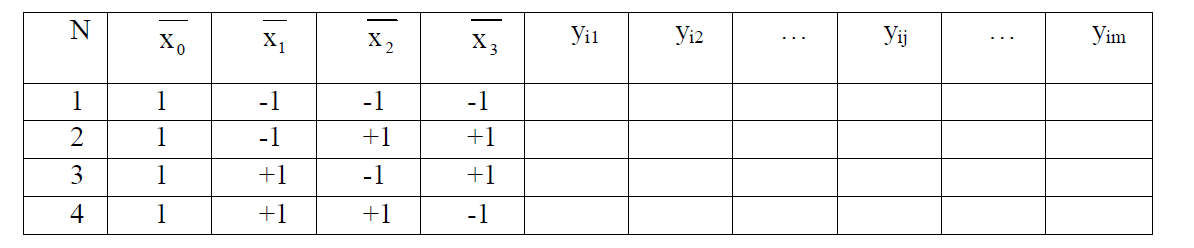


x1min = 10; x2min = 30; x3min = 10;

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

Ymin = 217 Ymax = 247

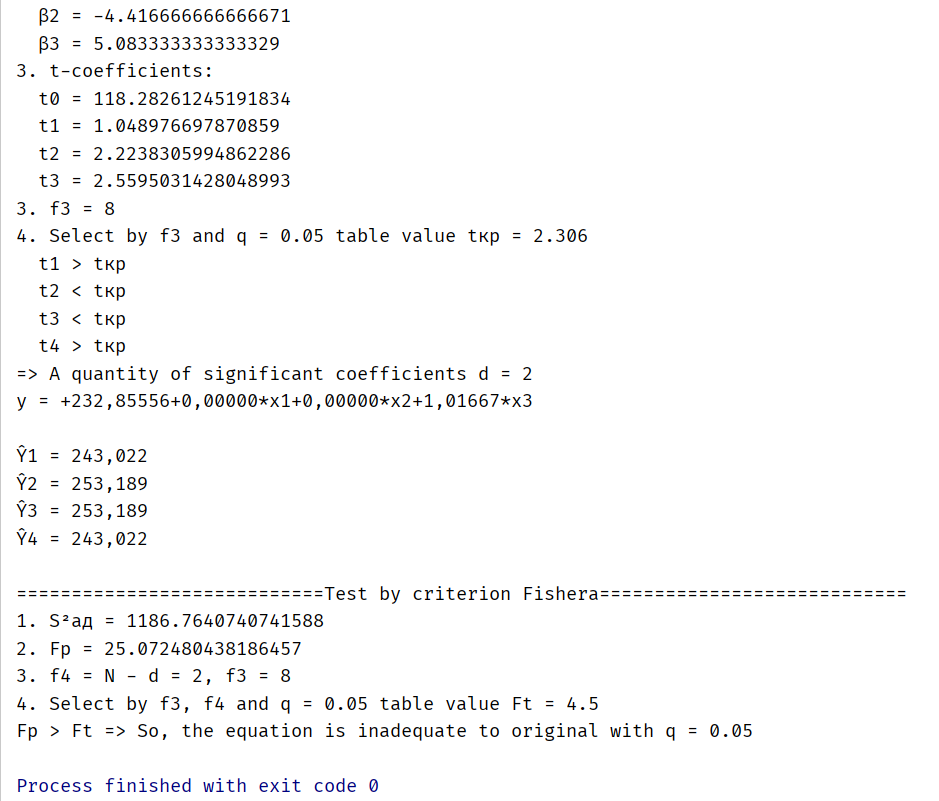
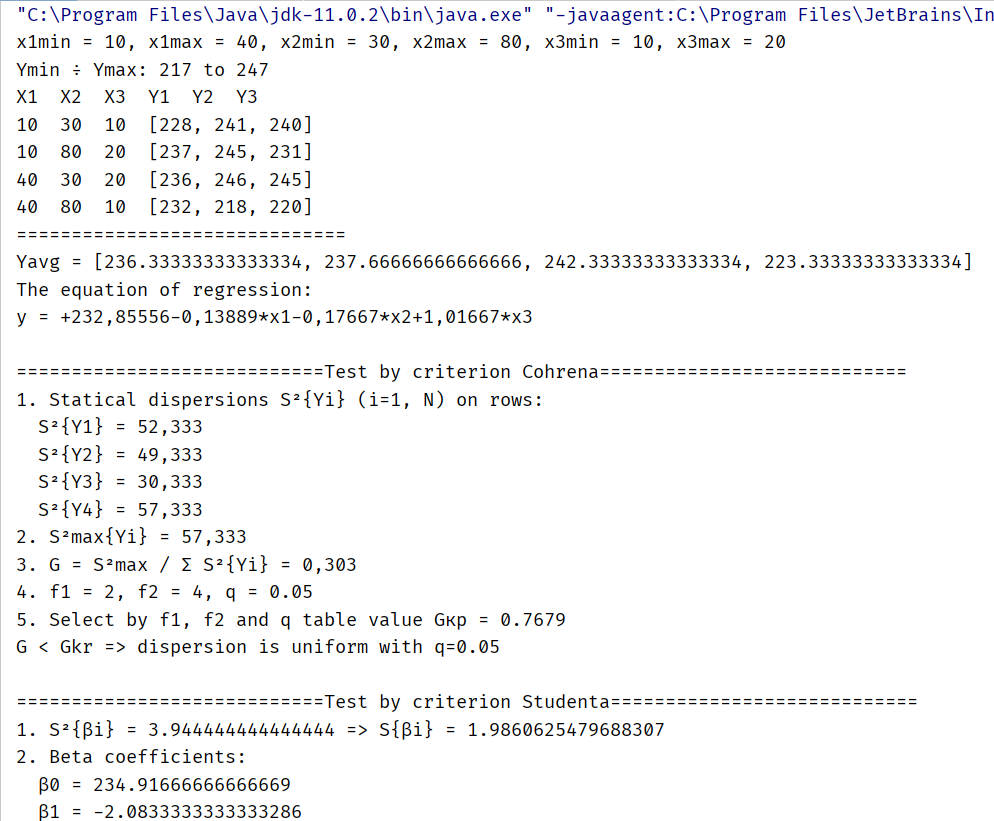
Нормована матриця планування:



1. Лістинг програми:

**package** lab3;  
  
**import** java.util.Arrays;  
**import** java.util.Random;  
  
**public class** ThreeFactorsExperiment {  
 **private int N** = 4;  
 **private int Ymin**, **Ymax**;  
 **private static int** *m* = 3;  
 **private int**[][] **xk** = {  
 {1, 1, 1, 1},  
 {-1, -1, 1, 1},  
 {-1, 1, -1, 1},  
 {-1, 1, 1, -1},  
 };  
 **private int**[][] **x**, **y**;  
 **private int**[] **X**;  
 **private double**[] **Yavg** = **new double**[4];  
 **private double**[] **b** = **new double**[4];  
 **private double**[] **S2y**, **y\_**;  
 **private int d**;  
  
 **public void** printMatrixOfPlanning() {  
 System.***out***.print(**"X1\tX2\tX3\t"**);  
 **for** (**int** i = 0; i < *m*; i++) {  
 System.***out***.printf(**"Y%d\t"**, i + 1);  
 }  
 System.***out***.println();  
 **for** (**int** i = 0, k = 0; i < 4; i++) {  
 **for** (**int** j = 0; j < **x**.**length**; j++) {  
 System.***out***.print(**x**[j][k] + **"\t"**);  
 }  
 System.***out***.print(Arrays.*toString*(**y**[i]) + **"\n"**);  
 k++;  
 }  
 System.***out***.println(**"=============================="**);  
 System.***out***.println(**"Yavg = "** + Arrays.*toString*(**Yavg**));  
 }  
  
 **public** ThreeFactorsExperiment(**int**[] X, **int** Ymin, **int** Ymax) {  
 **if** (X.**length** != 6) {  
 **throw new** RuntimeException(**"The length of array 'x' must be equaled 6! But founded "** + X.**length**);  
 }  
 **this**.**Ymin** = Ymin;  
 **this**.**Ymax** = Ymax;  
 **this**.**X** = X;  
 generateMatrixOfPlanning(*m*);  
 }  
  
 **private void** generateMatrixOfPlanning(**int** m) {  
 Random random = **new** Random();  
 **d** = 0;  
 **y** = **new int**[4][m];  
 **x** = **new int**[3][4];  
 **int** total = 0;  
  
 **for** (**int** i = 0, k = 0; i < **x**.**length**; i++, k += 2) {  
 **for** (**int** j = 0; j < **x**[i].**length**; j++) {  
 **x**[i][j] = (**xk**[i+1][j] == -1) ? **X**[k] : **X**[k + 1];  
 }  
 }  
  
 **for** (**int** i = 0; i < **y**.**length**; i++) {  
 **for** (**int** j = 0; j < **y**[i].**length**; j++) {  
 **y**[i][j] = **Ymin** + random.nextInt(**Ymax** - **Ymin** + 1);  
 total += **y**[i][j];  
 }  
 **Yavg**[i] = (**double**) total / m;  
 total = 0;  
 }  
 }  
  
 **public double** getDeterminant(**double**[][] a) {  
 **if** (a.**length** == 1)  
 **return** a[0][0];  
  
 **else** {  
 **double** res = 0;  
 **for** (**int** i = 0; i < a.**length**; i++) {  
 **double**[][] arr = **new double**[a.**length** - 1][a.**length** - 1];  
 **for** (**int** row = 1, v = 0; row < a.**length**; row++, v++) {  
 **for** (**int** col = 0, t = 0; col < a[row].**length**; col++) {  
 **if** (col != i) {  
 arr[v][t++] = a[row][col];  
 }  
 }  
 }  
 res += Math.*pow*(-1, i + 2) \* a[0][i] \* getDeterminant(arr);  
 }  
 **return** res;  
 }  
 }  
  
 **private double** findACoefficient(**int**[] a1, **int**[] a2) {  
 **double** s = 0;  
 **for** (**int** i = 0; i < a1.**length**; i++) {  
 s += (a1[i] \* a2[i]);  
 }  
 **return** s / a1.**length**;  
 }  
  
 **public void** findNormalizedCoefficients() {  
 **double**[] mx = **new double**[*m*]; *// mx1, mx2, mx3* **double**[] a = **new double**[*m*]; *// a1, a2, a3* **double**[] a\_ = **new double**[*m*]; *// a11, a22, a33* **double** my = Arrays.*stream*(**Yavg**).sum() / **Yavg**.**length**;  
  
 **for** (**int** i = 0; i < 3; i++) {  
 **double** sumMx = 0;  
 **double** sumA = 0;  
 **for** (**int** j = 0; j < **x**[i].**length**; j++) {  
 sumMx += **x**[i][j];  
 sumA += (**x**[i][j] \* **Yavg**[j]);  
 }  
 mx[i] = sumMx / 4;  
 a[i] = sumA / 4;  
 a\_[i] = findACoefficient(**x**[i], **x**[i]);  
 }  
  
 **double** a12 = findACoefficient(**x**[0], **x**[1]);  
 **double** a13 = findACoefficient(**x**[0], **x**[2]);  
 **double** a23 = findACoefficient(**x**[1], **x**[2]);  
  
 *//Finding a coefficients, using a method of Kramer* **double** delta = getDeterminant(**new double**[][]{  
 {1, mx[0], mx[1], mx[2]},  
 {mx[0], a\_[0], a12, a13},  
 {mx[1], a12, a\_[1], a23},  
 {mx[2], a13, a23, a\_[2]},  
 });  
  
 **double** delta1 = getDeterminant(**new double**[][]{  
 {my, mx[0], mx[1], mx[2]},  
 {a[0], a\_[0], a12, a13},  
 {a[1], a12, a\_[1], a23},  
 {a[2], a13, a23, a\_[2]},  
 });  
  
 **double** delta2 = getDeterminant(**new double**[][]{  
 {1, my, mx[1], mx[2]},  
 {mx[0], a[0], a12, a13},  
 {mx[1], a[1], a\_[1], a23},  
 {mx[2], a[2], a23, a\_[2]},  
 });  
  
 **double** delta3 = getDeterminant(**new double**[][]{  
 {1, mx[0], my, mx[2]},  
 {mx[0], a\_[0], a[0], a13},  
 {mx[1], a12, a[1], a23},  
 {mx[2], a13, a[2], a\_[2]},  
 });  
  
 **double** delta4 = getDeterminant(**new double**[][]{  
 {1, mx[0], mx[1], my},  
 {mx[0], a\_[0], a12, a[0]},  
 {mx[1], a12, a\_[1], a[1]},  
 {mx[2], a13, a23, a[2]},  
 });  
  
 **b**[0] = delta1 / delta;  
 **b**[1] = delta2 / delta;  
 **b**[2] = delta3 / delta;  
 **b**[3] = delta4 / delta;  
  
 System.***out***.printf(**"The equation of regression: \ny = %+.5f%+.5f\*x1%+.5f\*x2%+.5f\*x3\n\n"**, **b**[0], **b**[1], **b**[2], **b**[3]);  
 }  
  
 **public void** testByCriterionKohrena() {  
 **double** S2max = 0;  
 **double** q = 0.05;  
 **double**[][] CohrenaTable = {  
 {.9985, .9750, .9392, .9057, .8772, .8534, .8332, .8159, .8010, .7880},  
 {.9669, .8709, .7977, .7457, .7071, .6771, .6530, .6333, .6167, .6025},  
 {.9065, .7679, .6841, .6287, .5892, .5598, .5365, .5175, .5017, .4884},  
 {.8412, .6838, .5981, .5440, .5063, .4783, .4564, .4387, .4241, .4118},  
 {.7808, .6161, .5321, .4803, .4447, .4184, .3980, .3817, .3682, .3568},  
 {.7271, .5612, .4800, .4307, .3974, .3726, .3535, .3384, .3259, .3154},  
 {.6798, .5157, .4377, .3910, .3595, .3362, .3185, .3043, .2926, .2829},  
 {.6385, .4775, .4027, .3584, .3286, .3067, .2901, .2768, .2659, .2568},  
 {.6020, .4450, .3733, .3311, .3029, .2823, .2666, .2541, .2439, .2353},  
 };  
  
 System.***out***.println(**"============================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]);  
 }  
 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 / Σ S²{Yi} = %.3f\n"**,G);  
  
 **int** f1 = *m* - 1, f2 = **N**;  
 System.***out***.println(**"4. f1 = "**+f1+**", f2 = "**+f2+**", q = "**+q);  
  
 **double** Gkr = CohrenaTable[f2 - 2][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();  
 findNormalizedCoefficients();  
 testByCriterionKohrena();  
 }  
 }  
  
 **public void** testByStudentCriterion() {  
 System.***out***.println(**"\n============================Test by criterion Studenta============================"**);  
 **double**[] StudentaTable = {12.71, 4.303, 3.182, 2.776, 2.571, 2.447, 2.365, 2.306, 2.262,  
 2.228, 2.201, 2.179, 2.160, 2.145, 2.131, 2.12, 2.11, 2.101, 2.093, 2.086,  
 2.08, 2.074, 2.069, 2.064, 2.06, 2.056, 2.052, 2.048, 2.045, 2.042, 1.960  
 };  
 **double** S2beta = Arrays.*stream*(**S2y**).sum()/(**N**\***N**\**m*);  
 **double**[] beta = **new double**[**N**];  
 **double**[] t = **new double**[**N**];  
 **double** q = 0.05;  
  
 System.***out***.println(**"1. S²{βi} = "**+S2beta+**" => S{βi} = "**+Math.*sqrt*(S2beta));  
 System.***out***.println(**"2. Beta coefficients: "**);  
 **for** (**int** i = 0; i < **N**; i++) {  
 **double** sum = .0;  
 **for** (**int** j = 0; j < **xk**[i].**length**; j++) {  
 beta[i]+=(**Yavg**[j]\***xk**[i][j]);  
 }  
 beta[i] /= 4.0;  
 t[i] = Math.*abs*(beta[i]) / Math.*sqrt*(S2beta);  
 System.***out***.println(**" β"**+i+**" = "**+beta[i]);  
 }  
  
 System.***out***.println(**"3. t-coefficients: "**);  
 **for** (**int** i = 0; i < t.**length**; i++) {  
 System.***out***.println(**" t"**+i+**" = "**+t[i]);  
 }  
  
 **int** f3 = (*m*-1)\***N**;  
 **double** tkr = StudentaTable[f3-1];  
 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] = 0;  
 }  
 **else** {  
 System.***out***.printf(**" t%d > tкр\n"**, (i+1));  
 **d**++;  
 }  
 }  
 System.***out***.println(**"=> A quantity of significant coefficients d = "**+**d**);  
 System.***out***.printf(**"y = %+.5f%+.5f\*x1%+.5f\*x2%+.5f\*x3\n\n"**, **b**[0], **b**[1], **b**[2], **b**[3]);  
  
 **y\_** = **new double**[**N**];  
 **for** (**int** i = 0, k = 0; i < **N**; i++) {  
 **y\_**[i] = **b**[0]+**b**[1]\***x**[0][i]+**b**[2]\***x**[1][i]+**b**[3]\***x**[2][i];  
 System.***out***.printf(**"Ŷ%d = %.3f\n"**,i+1,**y\_**[i]);  
 }  
 }  
  
 **public void** testByFisheraCriterion() {  
 System.***out***.println(**"\n============================Test by criterion Fishera============================"**);  
 **double**[][] FisheraTable = {  
 {164.4, 199.5, 215.7, 224.6, 230.2, 234}, {18.5, 19.2, 19.2, 19.3, 19.3, 19.3},  
 {10.1, 9.6, 9.3, 9.1, 9, 8.9}, {7.7, 6.9, 6.6, 6.4, 6.3, 6.2},  
 {6.6, 5.8, 5.4, 5.2, 5.1, 5}, {6, 5.1, 4.8, 4.5, 4.4, 4.3},  
 {5.5, 4.7, 4.4, 4.1, 4, 3.9}, {5.3, 4.5, 4.1, 3.8, 3.7, 3.6},  
 {5.1, 4.3, 3.9, 3.6, 3.5, 3.4}, {5, 4.1, 3.7, 3.5, 3.3, 3.2},  
 {4.8, 4, 3.6, 3.4, 3.2, 3.1}, {4.8, 3.9, 3.5, 3.3, 3.1, 3},  
 {4.7, 3.8, 3.4, 3.2, 3, 2.9}, {4.6, 3.7, 3.3, 3.1, 3, 2.9},  
 {4.5, 3.7, 3.3, 3.1, 2.9, 2.8}, {4.5, 3.6, 3.2, 3, 2.9, 2.7},  
 {4.5, 3.6, 3.2, 3, 2.8, 2.7}, {4.4, 3.6, 3.2, 2.9, 2.8, 2.7},  
 {4.4, 3.5, 3.1, 2.9, 2.7, 2.6}, {4.4, 3.5, 3.1, 2.9, 2.7, 2.6},  
 };  
 **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+=(**y\_**[i] - **Yavg**[i])\*(**y\_**[i] - **Yavg**[i]);  
 }  
 **double** S2ad = (**double**) *m*/(**N**-**d**)\*sum;  
 System.***out***.println(**"1. S²ад = "**+S2ad);  
  
 **double** Fp = S2ad/(Arrays.*stream*(**S2y**).sum()/**N**); *// Fp = S²ад / S²в* System.***out***.println(**"2. Fp = "**+Fp);  
 System.***out***.println(**"3. f4 = N - d = "**+f4+**", f3 = "**+f3);  
  
 **double** Ft = FisheraTable[f3-1][f4-1];  
 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);  
 }  
}

1. Результат виконання роботи програми:



**Контрольні запитання**

1. *Що називається дробовим факторним експериментом?*

У деяких випадках немає необхідності проводити повний факторний експеримент (ПФЕ). Якщо буде використовуватися лінійна регресія, то можливо зменшити кількість рядків матриці ПФЕ до кількості коефіцієнтів регресійної моделі. Кількість дослідів слід скоротити, використовуючи для планування так звані регулярні дробові репліки від повного факторного експерименту, що містять відповідну кількість дослідів і зберігають основні властивості матриці планування – це означає дробовий факторний експеримент (ДФЕ).

1. *Для чого потрібно розрахункове значення Кохрена?*

Для перевірки дисперсії на однорідність.

1. *Для чого перевіряється критерій Стьюдента?*

Для перевірки значущості коефіцієнтів регресії. Тобто, Якщо виконується нерівність *ts*< *tтабл*, то приймається нуль-гіпотеза, тобто вважається, що знайдений коефіцієнт *βs* є статистично незначущим і його слід виключити з рівняння регресії. Якщо *ts* > *tтабл* то гіпотеза не підтверджується, тобто *βs* – значимий коефіцієнт і він залишається в рівнянні регресії.

1. *Чим визначається критерій Фішера і як його застосовувати?*

Отримане рівняння регресії необхідно перевірити на адекватність досліджуваному об'єкту. Для цієї мети необхідно оцінити, наскільки відрізняються середні значення у вихідної величини, отриманої в точках факторного простору, і значення у, отриманого з рівняння регресії в тих самих точках факторного простору. Для цього використовують дисперсію адекватності. Адекватність моделі перевіряють за F-критерієм Фішера, який дорівнює відношенню дисперсії адекватності до дисперсії відтворюваності.

**Висновок:** Отже, у ході виконання лабораторної роботи № 3 провели дробовий трьохфакторний експеримент. Склали матрицю планування, знайшли

коефіцієнти рівняння регресії, провели 3 статистичні перевірки. Була написана текстова програма, результати наведені вище. Результати співпадають із калькулятором. Кінцева мета роботи досягнута!