**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт №3. Системы управления, информатика и электроэнергетика. Кафедра №304 Вычислительные машины, системы и комплексы

Отчет по лабораторной работе

по учебной дисциплине

теория оптимального планирования и управления

на тему

**«Методы одномерной безусловной оптимизации»**

Группа М30-207Б

Выполнили студенты:

Кривонос А.А.

Приняли:

Татарникова Е.М.

Давыдкина Е.А.

Москва, 2019г.

Содержание**:**

1. Задание………………………………………………………………......... 2

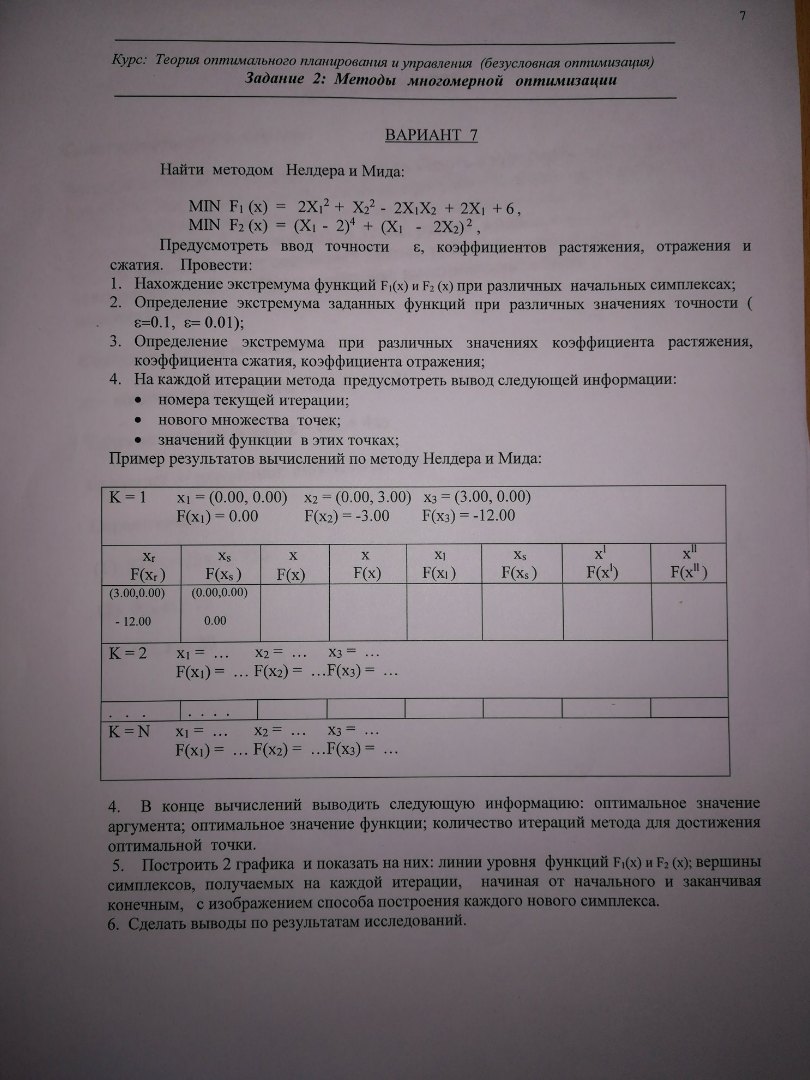
2. Текст программы……….………………………………………………… 3

3. Графики …….……...................................................................................... 9

4. Тесты……………………………………………….................................... 11

5. Вывод………………….……………………………….............................. 18

**Задание**



**Текст программы**

#include <iostream>

#include <iomanip>

using namespace std;

const int i = 2; //колличество переменных

const int j = i + 1; //колличество вершин

//Функции

//

double F(double x[2]) { return (2\*x[0] \* x[0] + x[1] \* x[1] -2\*x[0] \* x[1] +2\*x[0] +6); }

//

//double F(double x[2]) { return (x[0]) \* (x[0]) + x[1] \* x[1]; }

//

//double F(double x[2]) {

//return ((x[0] - x[1]) \* (x[0] - x[1]) + 7 \* (x[1] \* x[1] - x[0]) \* (x[1] \*

//x[1] - x[0]));

//}

//----------------------------------------------------------------------------------//

//Условие остановки (дисперсия)

double D(double xh[2], double xg[2], double xl[2], double xc[2])

{

return sqrt(1 / (j + 1) \* ((F(xh)) - F(xc)) \* ((F(xh)) - F(xc)) + ((F(xg)) - F(xc)) \* ((F(xg)) -

F(xc)) + ((F(xl)) - F(xc)) \* ((F(xl)) - F(xc)));

}

void сорт(double x[j][i]);

void Таблица(double x[j][i], double х\_наим[2], double х\_2\_по\_велечене[2], double

х\_наиб[2], double центр\_тяж[2], double отр[2], double отр\_рас[2], double

отр\_сжат[2], int k);

int main() {

system("color F0"); //фон белый буквы черные

int n; //индексная переменная

int k = 0; //количество итераций

bool f = 0; //вспомогательный флаг

double Temp[i]; //обменная переменная

double Eps = 0.01; //точность

double a = 1; //коэфицент отражения

double b = 0.5; //коэфицент сжатия

double y = 2; //коэфицент рястяжения

//начальный набор точек

double х\_наиб[2]; //точка с наибольшем значением функции

double х\_2\_по\_велечене[2]; //точка с 2 по велечине значением функции

double х\_наим[2]; //точка с самым маленьким значением функции

double центр\_тяж[2]; //центр тяжести

double отр[2]; //отраженный х

double отр\_рас[2]; //отраженный х c растяжением

double отр\_сжат[2]; //отраженный х с жатеем

double x[j][i] = { { -2,-2 },{ -1,-1 },{ 0,-1 } };

setlocale(LC\_ALL, "Russian");

cout << "Введите Eps : ";

cin >> Eps;

cout << "Введите коэфицент отражения : ";

cin >> a;

cout << "Введите коэфицент сжатия : ";

cin >> b;

cout << "Введите коэфицент рястяжения : ";

cin >> y;

setlocale(LC\_ALL, "C");

cout << "\n\n" << char(218) << setfill(char(196)) << setw(154) << char(191) << endl;

do {

k++;

сорт(x); //сортировка точек

for (n = 0; n < i; n++)

{

центр\_тяж[n] = 0;

отр[n] = 0;

отр\_рас[n] = 0;

отр\_сжат[n] = 0;

х\_наиб[n] = x[i][n];

х\_2\_по\_велечене[n] = x[i - 1][n];

х\_наим[n] = x[0][n];

}

f = 0;

//поиск центра тяжести

for (n = 0; n < i; n++)

for (int m = 0; m < i; m++)

центр\_тяж[n] = центр\_тяж[n] + (x[m][n]);

центр\_тяж[0] = центр\_тяж[0] / i; центр\_тяж[1] = центр\_тяж[1] / i;

for (n = 0; n < i; n++)

отр[n] = центр\_тяж[n] + a \* (центр\_тяж[n] - х\_наиб[n]);

if (F(отр) <= F(х\_наим))

{

for (n = 0; n < i; n++)

отр\_рас[n] = центр\_тяж[n] + y \* (отр[n] - центр\_тяж[n]);

double x7 = F(отр\_рас);

if (F(отр\_рас) < F(отр))

for (n = 0; n < i; n++)

х\_наиб[n] = отр\_рас[n];

else

for (n = 0; n < i; n++)

х\_наиб[n] = отр[n];

}

else if (F(х\_наим) <= F(отр) && F(отр) <= F(х\_2\_по\_велечене))

{

for (n = 0; n < i; n++)

х\_наиб[n] = отр[n];

}

else if (F(х\_2\_по\_велечене) <= F(отр) && F(отр) <= F(х\_наиб))

{

for (n = 0; n < i; n++)

{

Temp[n] = отр[n];

отр[n] = х\_наиб[n];

х\_наиб[n] = Temp[n];

f = 1;

}

}

else if (F(х\_наиб) <= F(отр))

f = 1;

if (f) {

for (n = 0; n < i; n++)

отр\_сжат[n] = центр\_тяж[n] + b \* (х\_наиб[n] - центр\_тяж[n]);

if (F(отр\_сжат) < F(х\_наиб))

for (n = 0; n < i; n++)

х\_наиб[n] = отр\_сжат[n];

if (F(отр\_сжат) >= F(х\_наиб))

{

for (n = 0; n < i; n++)

{

x[i][n] = х\_наиб[n];

x[i - 1][n] = х\_2\_по\_велечене[n];

x[0][n] = х\_наим[n];

}

сорт(x); //сортировка точек

for (n = 0; n < i; n++)

{

х\_наиб[n] = x[i][n];

х\_2\_по\_велечене[n] = x[i - 1][n];

х\_наим[n] = x[0][n];

}

for (n = 0; n < i; n++)

{

х\_наиб[n] = (х\_наиб[n] + х\_наим[n]) / 2;

х\_2\_по\_велечене[n] = (х\_2\_по\_велечене[n] + х\_наим[n]) / 2;

}

}

}

for (n = 0; n < i; n++)

{

x[i][n] = х\_наиб[n];

x[i - 1][n] = х\_2\_по\_велечене[n];

x[0][n] = х\_наим[n];

}

Таблица(x, х\_наим, х\_2\_по\_велечене, х\_наиб, центр\_тяж, отр, отр\_рас, отр\_сжат, k);

} while (fabs(D(х\_наиб, х\_2\_по\_велечене, х\_наим, центр\_тяж)) > Eps);

сорт(x); //сортировка точек

cout << char(179) << " k = " << left << setfill(char(32)) << setw(17) << k << right

<< "x\* = (" << setfill(char(32)) << setw(7) << round(x[0][0] \* 100) / 100

<< " : " << left << setfill(char(32)) << setw(7) << round(x[0][1] \* 100) / 100 << right

<< ") " << "f(x\*) = " << left << setfill(char(32)) << setw(21)

<< round(F(x[0]) \* 100) / 100 << right << setfill(char(32)) << setw(78)

<< char(179) << endl;

cout << char(192) << setfill(char(196)) << setw(154) << char(217) << "\n\n";

system("pause");

}

void сорт(double x[j][i])

{

double Min; //минимальный элемент

int jMin; //индекс минимального элемента

int iSort; //граница отсортированной области

int k; //индексная переменная

double Temp[i]; //обменная переменная

for (iSort = 0; iSort < j - 1; iSort++)

{

//первый элемент из неупорядоченных назначаем минимальным

Min = F(x[iSort]); //минимум

jMin = iSort; //его индекс

//ищем минимальный элемент в оставшейся части массива

for (k = iSort + 1; k < j; k++)

{

if (F(x[k]) < Min) //очередной кандидат на минимальный

{

//запоминаем минимальный элемент и его номер

Min = F(x[k]);

jMin = k;

}//if

}//for j

//нашли минимум в неупорядоченной части массива

//ставим его на место первого в неупорядоченной части массива

//меняем элементы местами

for (k = 0; k < i; k++)

{

Temp[k] = x[iSort][k];

x[iSort][k] = x[jMin][k];

x[jMin][k] = Temp[k];

}

} //for iSort

}

void Таблица(double x[j][i], double х\_наим[2], double х\_2\_по\_велечене[2], double

х\_наиб[2], double центр\_тяж[2], double отр[2], double отр\_рас[2], double

отр\_сжат[2], int k)

{

cout << char(179) << " k = " << left << setfill(char(32)) << setw(17) << k << right

<< "x1 = (" << setfill(char(32)) << setw(7) << round(x[0][0] \* 100) / 100 << " : "

<< left << setfill(char(32)) << setw(7) << round(x[0][1] \* 100) / 100 << right << ") "

<< "x2 = (" << setfill(char(32)) << setw(7) << round(x[1][0] \* 100) / 100 << " : "

<< left << setfill(char(32)) << setw(7) << round(x[1][1] \* 100) / 100 << right << ") "

<< "x3 = (" << setfill(char(32)) << setw(7) << round(x[2][0] \* 100) / 100 << " : "

<< left << setfill(char(32)) << setw(7) << round(x[2][1] \* 100) / 100 << right << ") "

<< setfill(char(32)) << setw(57) << char(179) << endl;

cout << char(179) << setfill(char(32)) << setw(30) << "f(x1) = " << left

<< setfill(char(32)) << setw(21) << round(F(х\_наим) \* 100) / 100 << right

<< "f(x2) = " << left << setfill(char(32)) << setw(21)

<< round(F(х\_2\_по\_велечене) \* 100) / 100 << right << "f(x3) = "

<< left << setfill(char(32)) << setw(21) << round(F(х\_наиб) \* 100) / 100 << right

<< setfill(char(32)) << setw(45) << char(179) << endl;

cout << char(195);

cout << setfill(char(196)) << setw(22);

for (int i = 0; i < 6; i++)

{

cout << char(194) << setfill(char(196)) << setw(22);

}

cout << char(180) << endl;

cout << char(179) << " xl "

<< char(179) << " xg "

<< char(179) << " xh "

<< char(179) << " xc "

<< char(179) << " xr "

<< char(179) << " xe "

<< char(179) << " xs "

<< char(179) << endl;

cout << char(179) << " f(xl) "

<< char(179) << " f(xg) "

<< char(179) << " f(xh) "

<< char(179) << " f(xc) "

<< char(179) << " f(xr) "

<< char(179) << " f(xe) "

<< char(179) << " f(xs) "

<< char(179) << endl;

cout << char(195);

cout << setfill(char(196)) << setw(22);

for (int i = 0; i < 6; i++)

{

cout << char(197) << setfill(char(196)) << setw(22);

}

cout << char(180) << endl;

/\*

double\* arrays[] = { х\_наим, х\_2\_по\_велечене, х\_наиб, центр\_тяж, отр,

отр\_рас, отр\_сжат };

int arrays\_size = sizeof(arrays)/sizeof(arrays[0]);\*/

/\*for (int i = 0; i < arrays\_size; ++i) {

cout << char(179) << " ("

<< fixed << setprecision(2) << arrays[i][0] << "; " << left

<< arrays[i][1] << right << ") ";

}

cout << endl;\*/

cout << char(179) << " (" << setfill(char(32)) << setw(7) << round(х\_наим[0] \* 100) /

100 << " ; " << left << setfill(char(32)) << setw(7) << round(х\_наим[1] \* 100) / 100 << right

<< ") ";

cout << char(179) << " (" << setfill(char(32)) << setw(7) << round(х\_2\_по\_велечене[0]

\* 100) / 100 << " ; " << left << setfill(char(32)) << setw(7) << round(х\_2\_по\_велечене[1] \*

100) / 100 << right << ") ";

cout << char(179) << " (" << setfill(char(32)) << setw(7) << round(х\_наиб[0] \* 100) /

100 << " ; " << left << setfill(char(32)) << setw(7) << round(х\_наиб[1] \* 100) / 100 << right

<< ") ";

cout << char(179) << " (" << setfill(char(32)) << setw(7) << round(центр\_тяж[0] \* 100)

/ 100 << " ; " << left << setfill(char(32)) << setw(7) << round(центр\_тяж[1] \* 100) / 100 <<

right << ") ";

cout << char(179) << " (" << setfill(char(32)) << setw(7) << round(отр\_рас[0] \* 100) / 100

<< " ; " << left << setfill(char(32)) << setw(7) << round(отр\_рас[1] \* 100) / 100 << right << ") ";

cout << char(179) << " (" << setfill(char(32)) << setw(7) << round(отр\_рас[0] \* 100) /

100 << " ; " << left << setfill(char(32)) << setw(7) << round(отр\_рас[1] \* 100) / 100 << right

<< ") ";

cout << char(179) << " (" << setfill(char(32)) << setw(7) << round(отр\_сжат[0] \* 100) /

100 << " ; " << left << setfill(char(32)) << setw(7) << round(отр\_сжат[1] \* 100) / 100 << right

<< ") ";

cout << char(179) << endl;

cout << char(179) << setfill(char(32)) << setw(13) << round(F(х\_наим) \* 100) / 100 <<

" ";

cout << char(179) << setfill(char(32)) << setw(13) << round(F(х\_2\_по\_велечене) \*

100) / 100 << " ";

cout << char(179) << setfill(char(32)) << setw(13) << round(F(х\_наиб) \* 100) / 100 <<

" ";

cout << char(179) << setfill(char(32)) << setw(13) << round(F(центр\_тяж) \* 100) / 100

<< " ";

cout << char(179) << setfill(char(32)) << setw(13) << round(F(отр) \* 100) / 100 << " ";

cout << char(179) << setfill(char(32)) << setw(13) << round(F(отр\_рас) \* 100) / 100 <<

" ";

cout << char(179) << setfill(char(32)) << setw(13) << round(F(отр\_сжат) \* 100) / 100

<< " ";

cout << char(179) << endl;

cout << char(195);

cout << setfill(char(196)) << setw(22);

for (int i = 0; i < 6; i++)

{

cout << char(193) << setfill(char(196)) << setw(22);

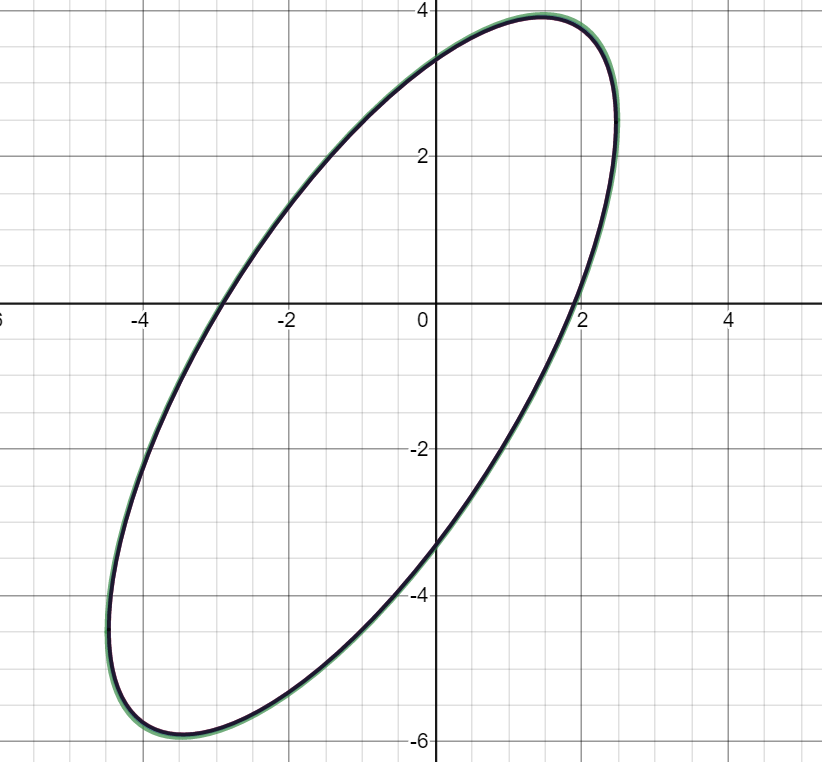
}

cout << char(180) << endl;

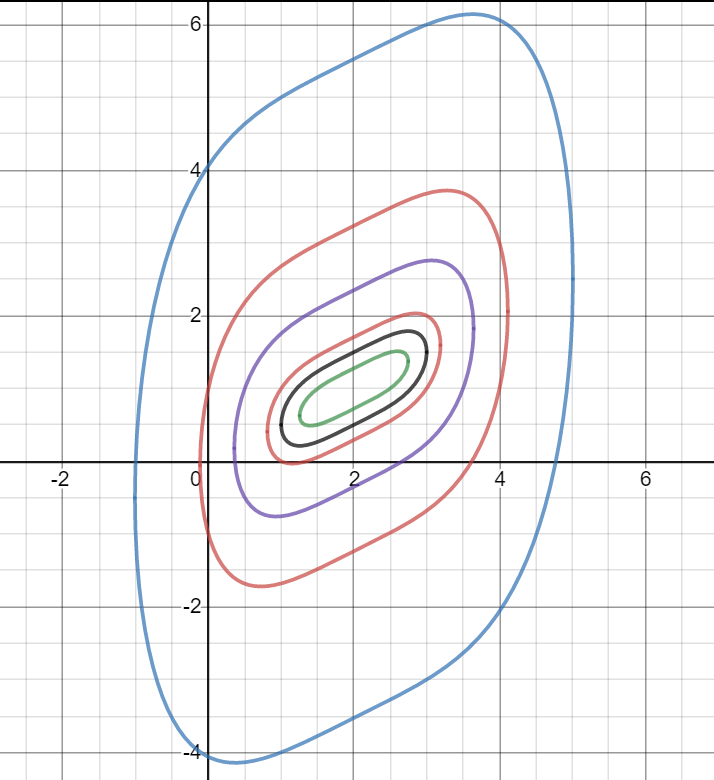
}

**Графики исходных функций**

1. Функция F1=2X12 + X22 - 2X1X2 +2X1 + 6

****

1. Функция F1=(X1-2)4+(X1-2X2)2

****

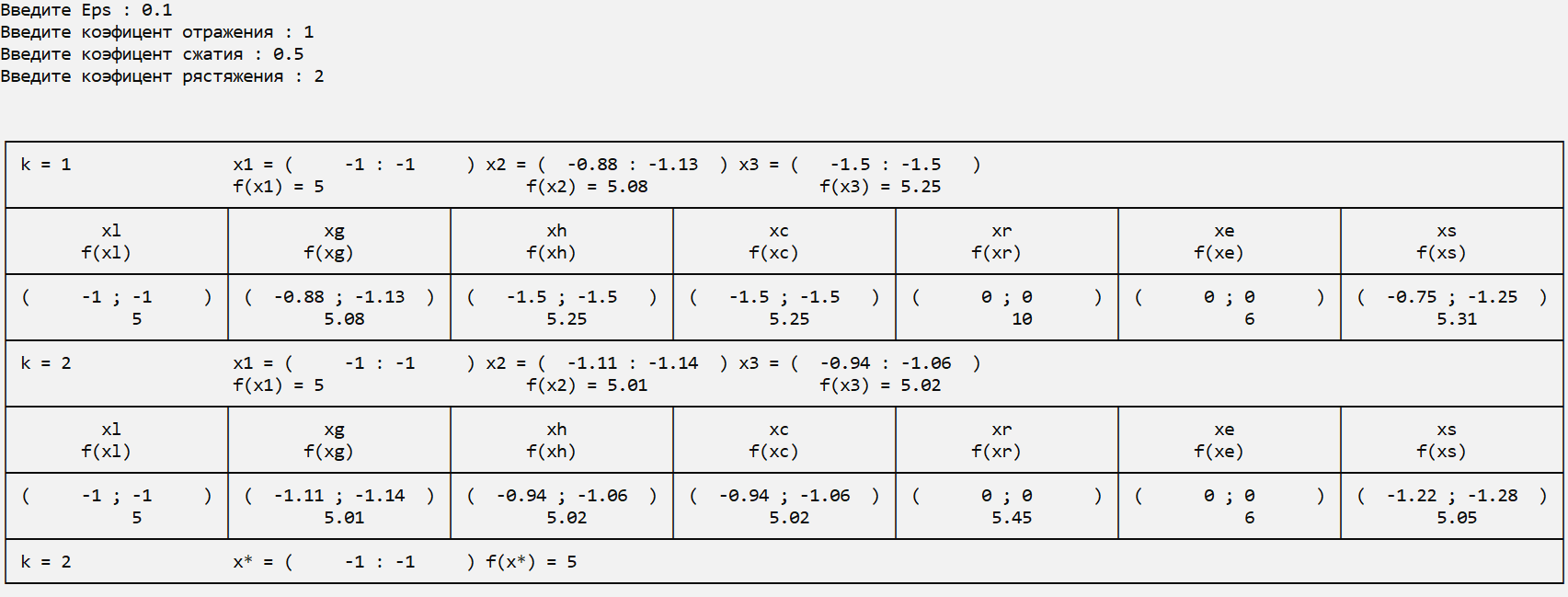
**Тесты**

1. Функция F1=2X12 + X22 - 2X1X2 +2X1 + 6

Коэффициент Eps= 0.1

Коэффициент отражения = 1

Коэффициент сжатия = 0.5

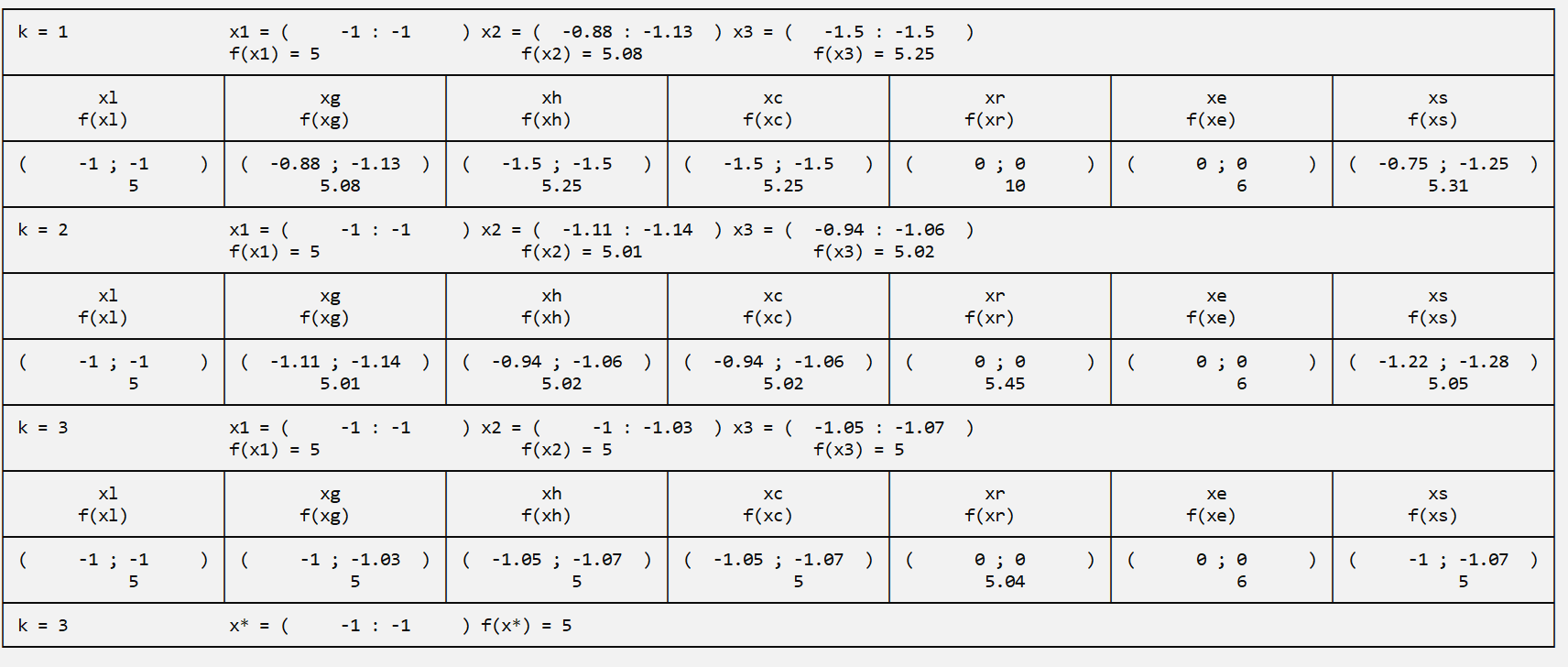
Коэффициент растяжения = 2

1. Функция F1=2X12 + X22 - 2X1X2 +2X1 + 6

Коэффициент Eps= 0.01

Коэффициент отражения = 1

Коэффициент сжатия = 0.5

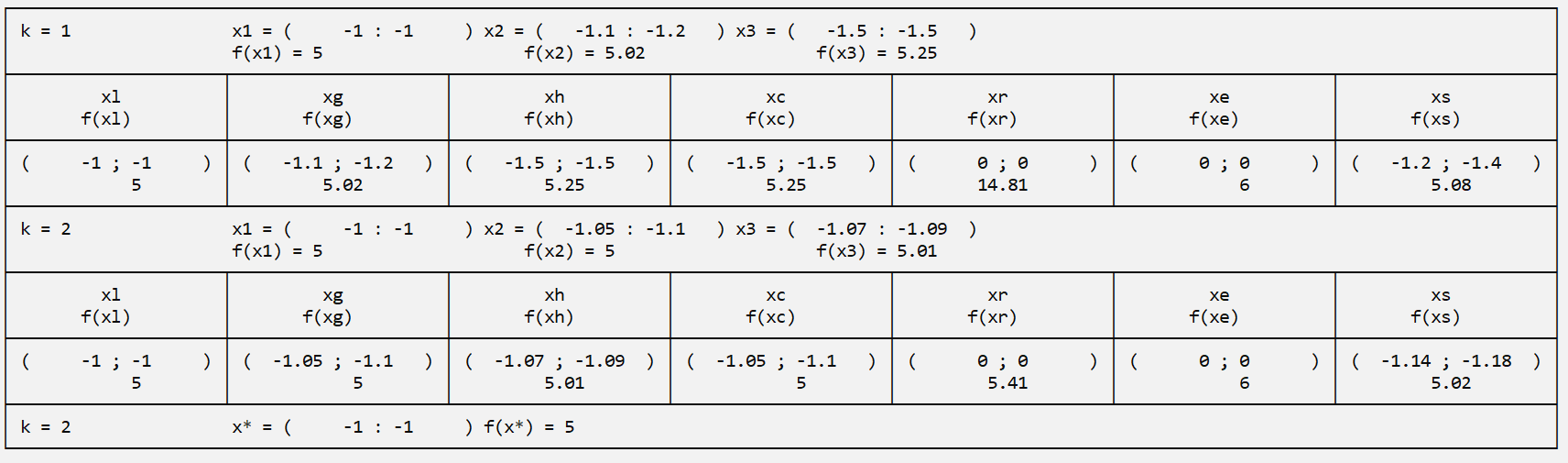
Коэффициент растяжения = 2

1. Функция F1=2X12 + X22 - 2X1X2 +2X1 + 6

Коэффициент Eps= 0.1

Коэффициент отражения = 1.5

Коэффициент сжатия = 0.2

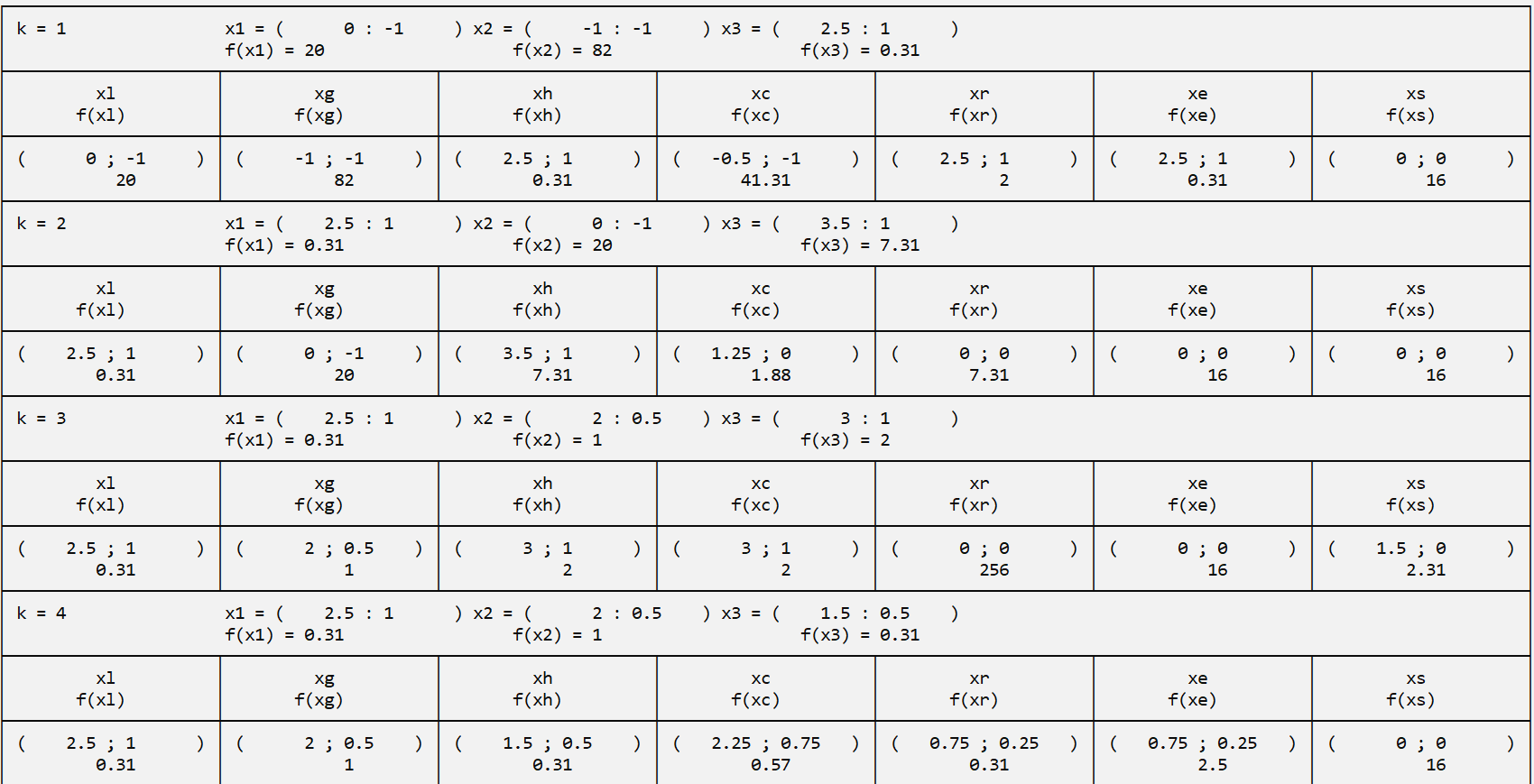
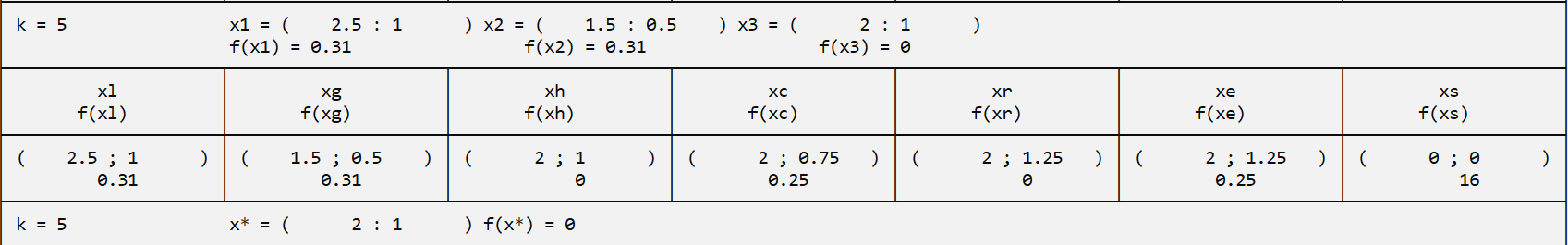
Коэффициент растяжения = 3

1. Функция F2=(X1-2)4+(X1-2X2)2

Коэффициент Eps= 0.1

Коэффициент отражения = 1

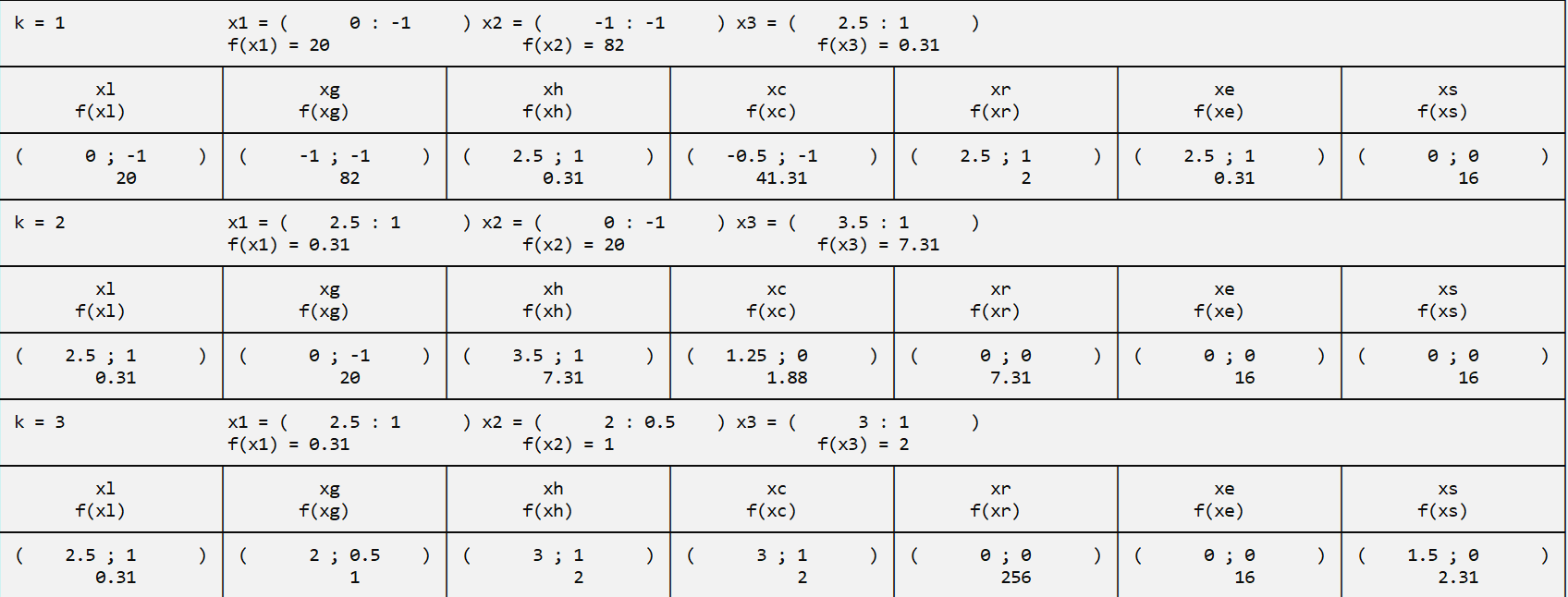
Коэффициент сжатия = 0.5

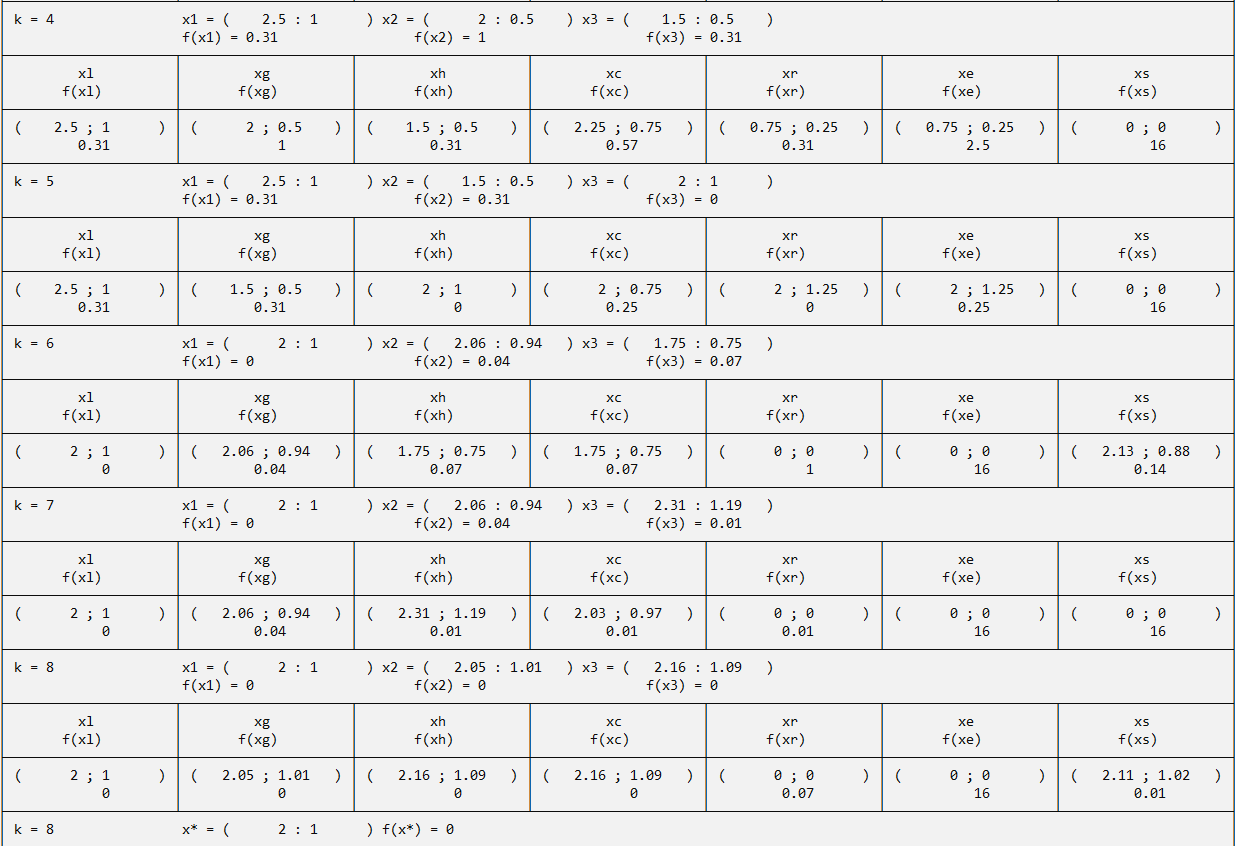
Коэффициент растяжения = 2

1. Функция F2=(X1-2)4+(X1-2X2)2

Коэффициент Eps= 0.01

Коэффициент отражения = 1

Коэффициент сжатия = 0.5 Коэффициент растяжения = 2



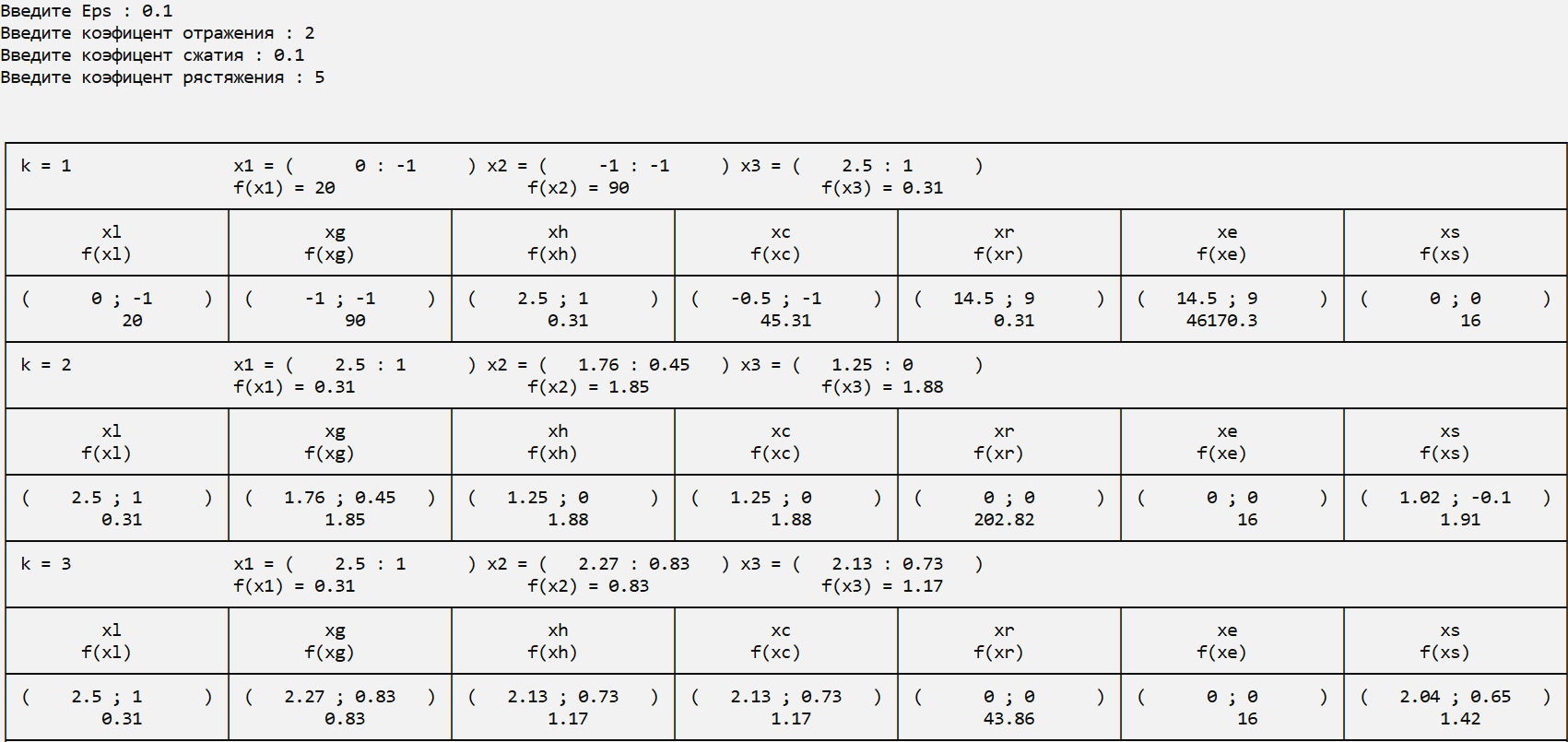
1. Функция F2=(X1-2)4+(X1-2X2)2

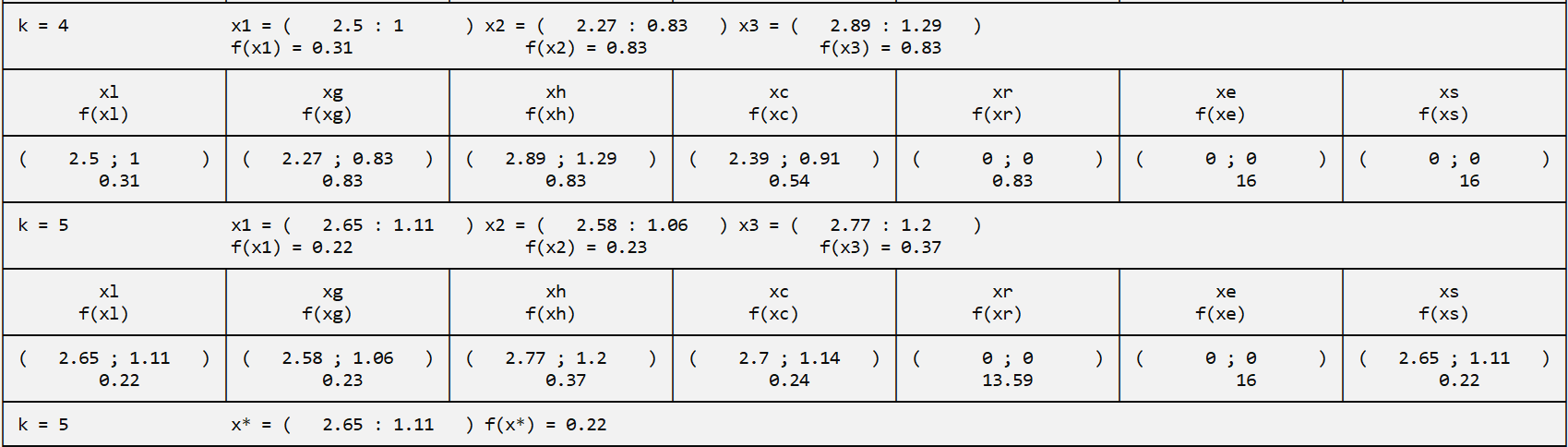
Коэффициент Eps= 0.1

Коэффициент отражения = 1.5

Коэффициент сжатия = 0.2

Коэффициент растяжения = 3





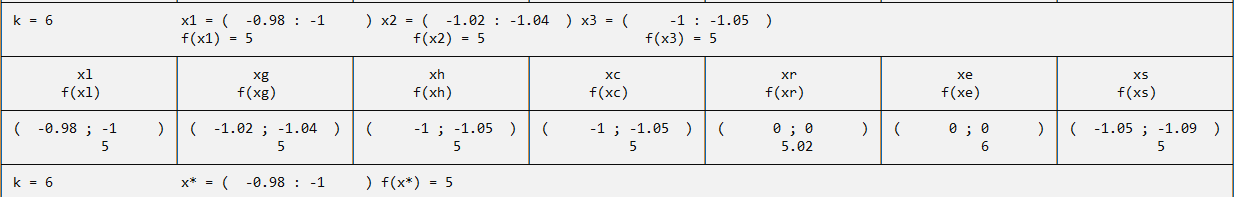
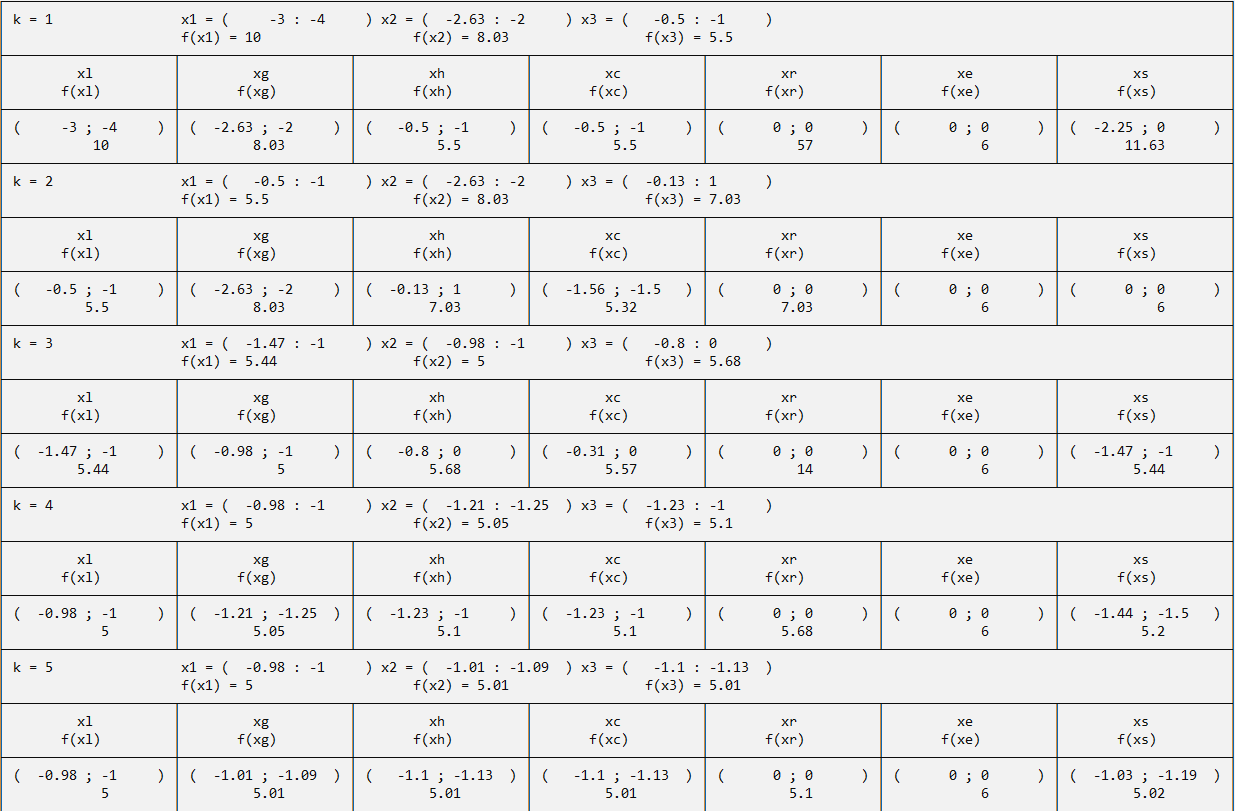
**Другой набор точек для функций**

1. Функция F1=2X12 + X22 - 2X1X2 +2X1 – 6

Коэффициент Eps= 0.1

Коэффициент отражения = 1

Коэффициент сжатия = 0.5

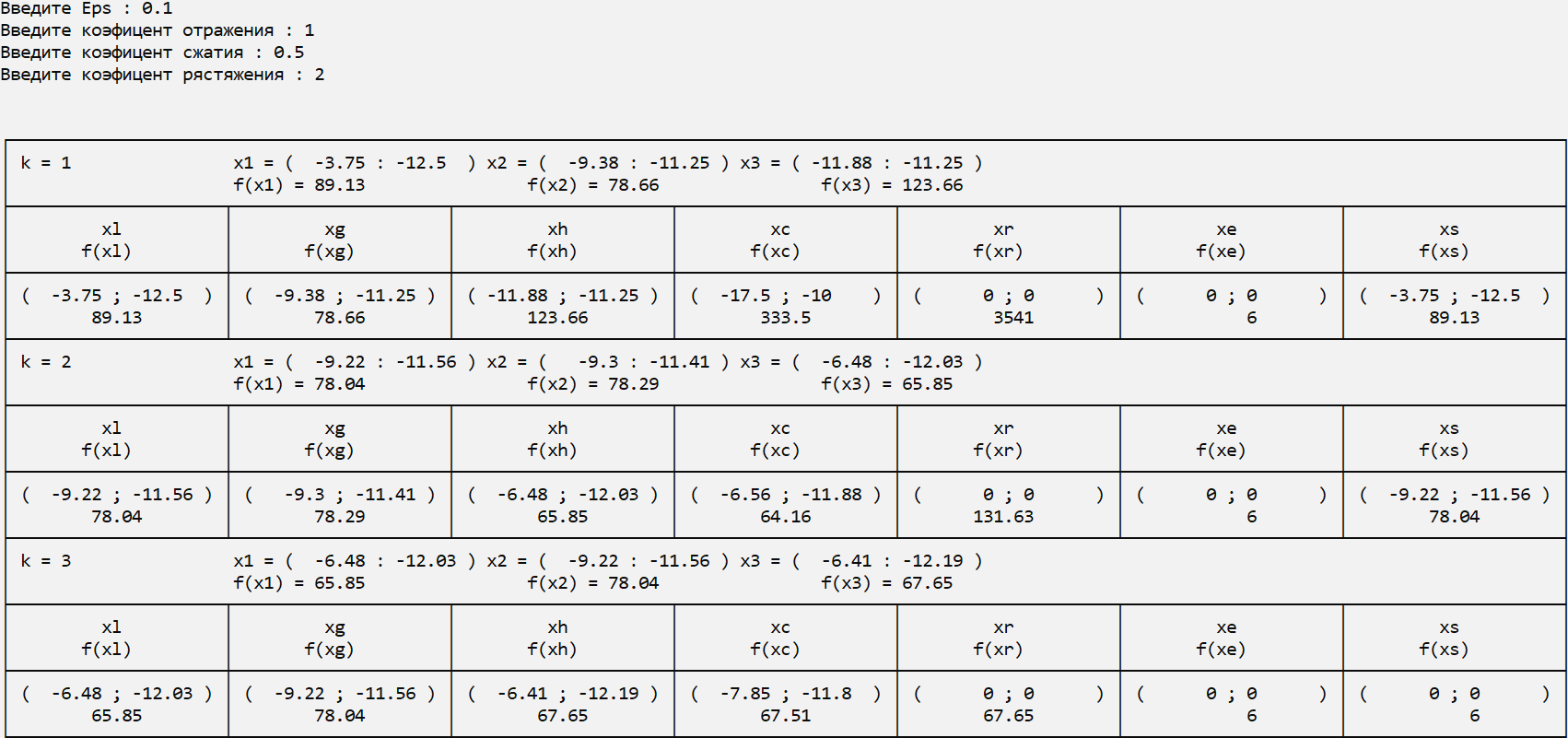
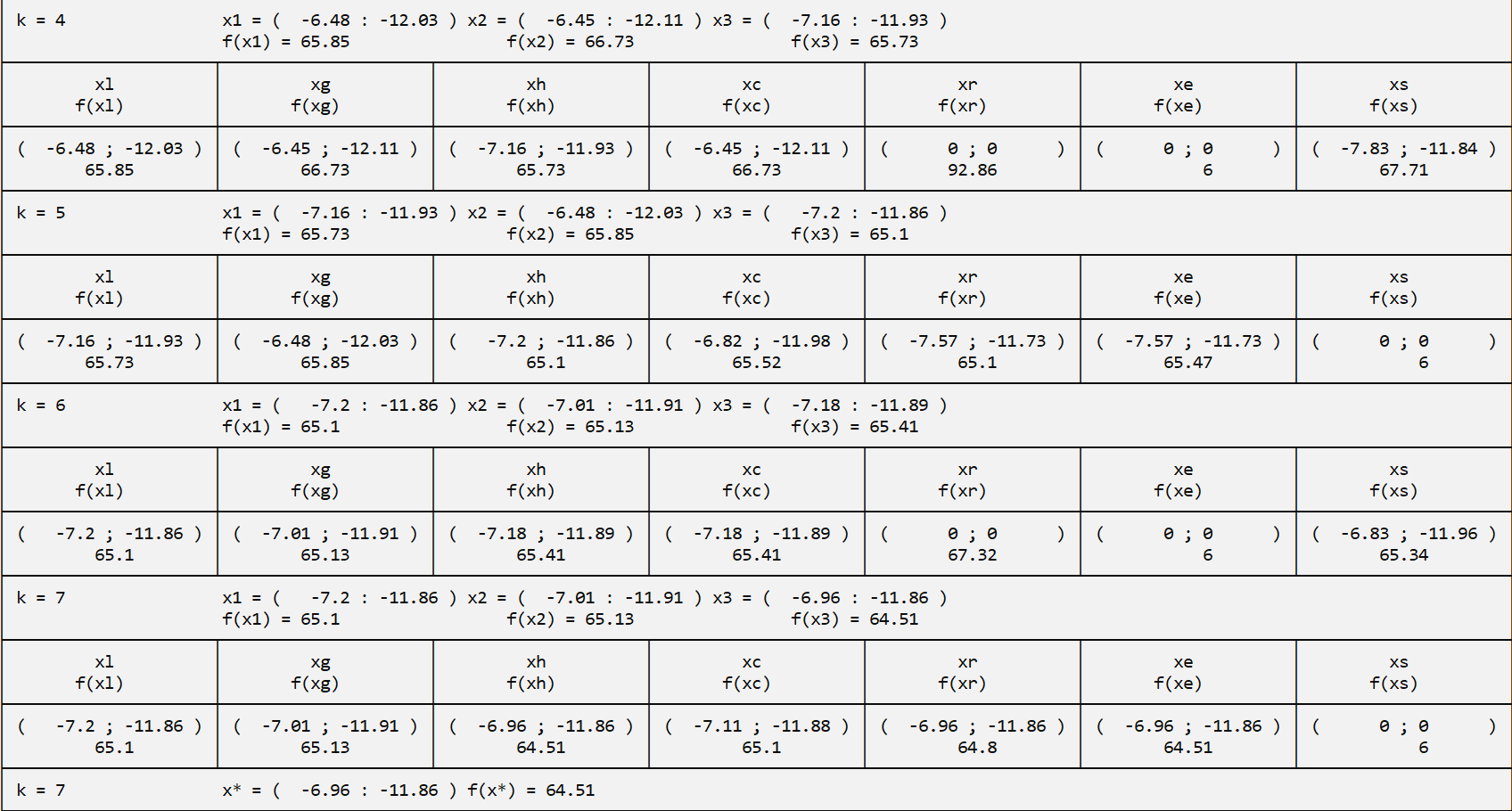
****Коэффициент растяжения = 2

1. Функция F1=2X12 + X22 - 2X1X2 +2X1 – 6

Коэффициент Eps= 0.1

Коэффициент отражения = 1

Коэффициент сжатия = 0.5

Коэффициент растяжения = 2

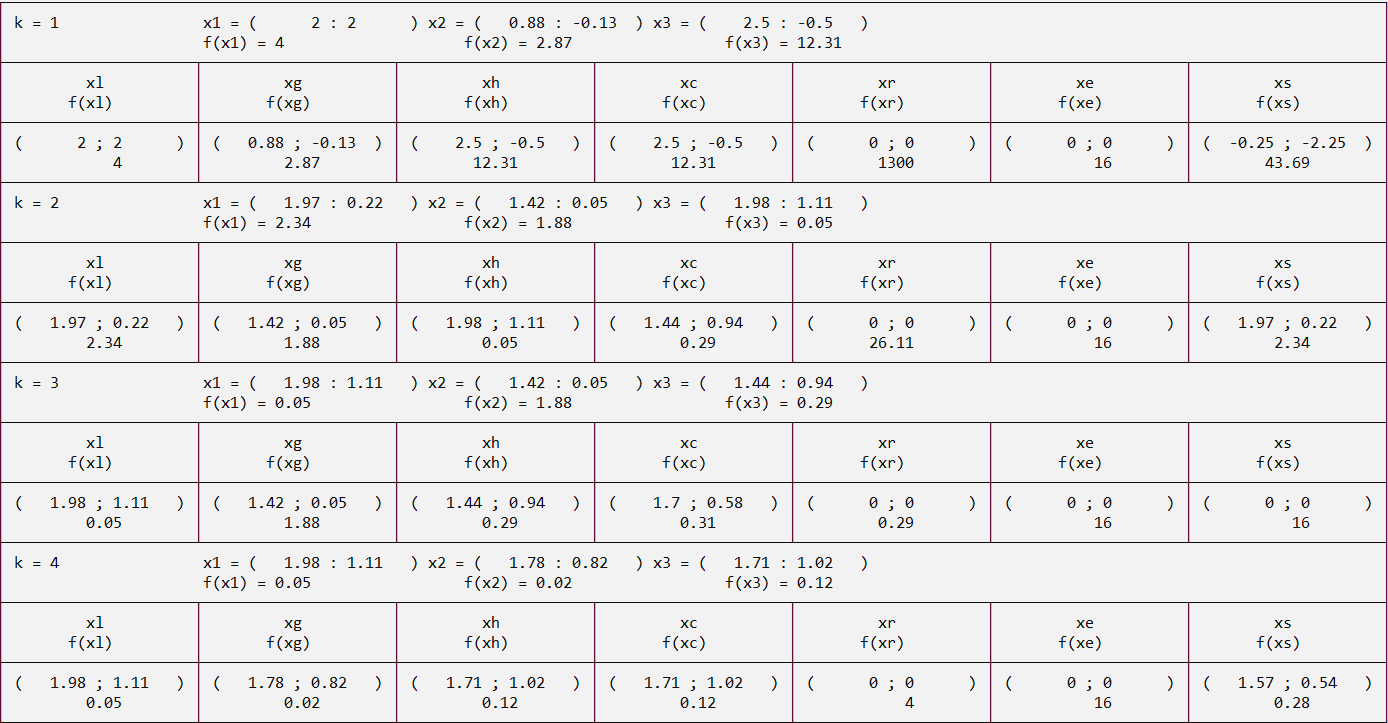
1. Функция F2=(X1-2)4+(X1-2X2)2

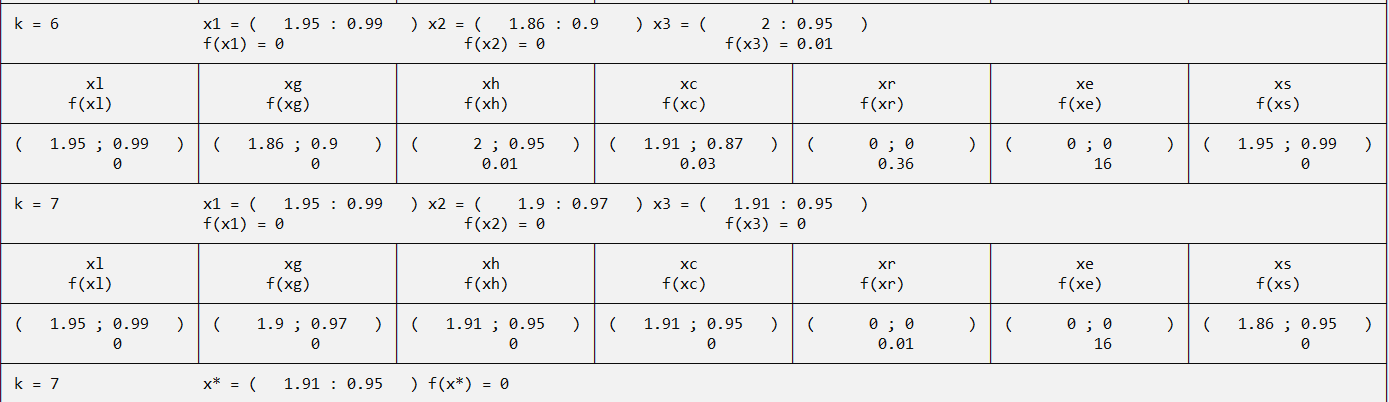
Коэффициент Eps= 0.01

Коэффициент отражения = 1

Коэффициент сжатия = 0.5

Коэффициент растяжения = 2





**Вывод**

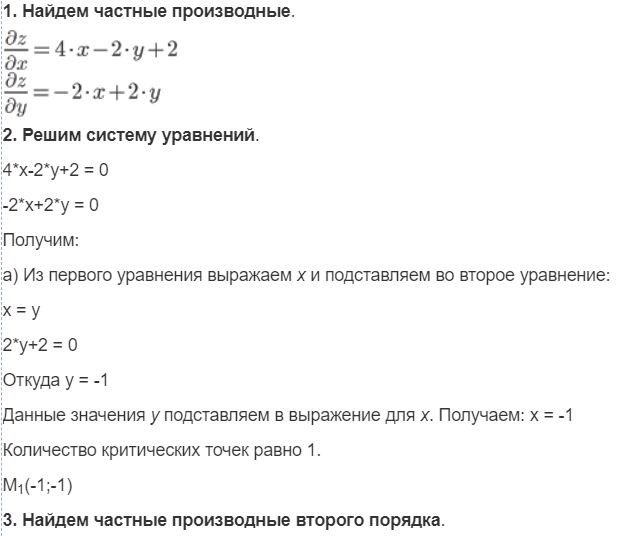
1) При повышении точности, если мы сразу угадываем оптимальную точку, количество итераций почти не меняется (в тесте №2 точность в 10 раз выше, чем в тесте №1, однако итераций больше на 1); если мы возьмем точку много дальше от оптимальной, то точность влияет на количество итераций при остальных одинаковых параметрах (в тесте №4 всего 5 итераций, а в тесте №5 всего 8 итераций)

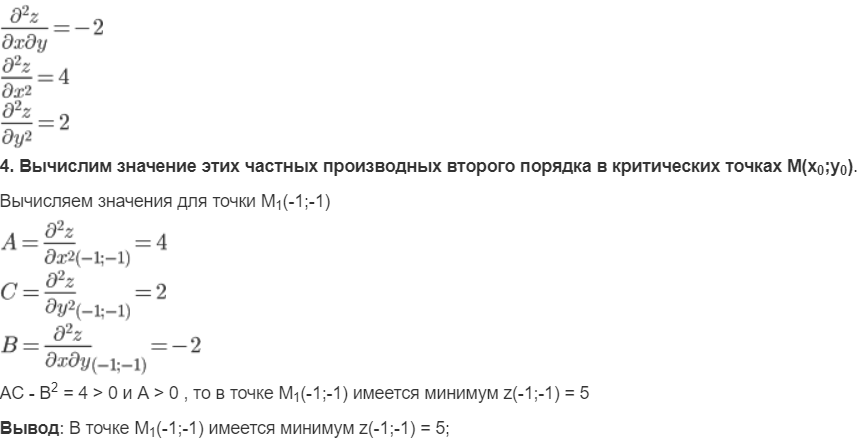
2) Вводимая точность не соответствует абсолютной, так как она используется

при подсчёте дисперсии точек а не сравнении с точным значением.

3) Следует использовать рекомендованные коэффициенты, так как они дают наиболее правдоподобный результат. Можно сравнить значение, полученное в программе, и значение, посчитанное с помощью производных. Любые изменения коэффициентов сжатия, растяжения и отражения приводят к потере точности результатов. (тест №4: Х\*=(2;1), F(X\*)=(0); тест №6: Х\*=(2,65;1,11), F(X\*)=(0,22); Значения полученные с помощью производных: Х\*=(2;1), F(X\*)=(0))

**Расчет оптимальных точек, с помощью производных**

Для F1=2X12 + X22 - 2X1X2 +2X1 + 6



Для F2=(X1-2)4+(X1-2X2)2

