**Метод Дихотомии**

Перед применением метода для поиска корней функции необходимо отделить корни одним из известных способов, например, графическим методом. Отделение корней необходимо в случае, если неизвестно на каком отрезке нужно искать корень.

Будем считать, что корень t функции f(x)=0 отделён на отрезке [a,b]. Задача заключается в том, чтобы найти и уточнить этот корень методом половинного деления. Другими словами, требуется найти приближённое значение корня с заданной точностью \eps.

Пусть функция f непрерывна на отрезке [a,b],

f(a)\cdot f(b)<0, \; \eps=0,01 и t\in[a,b] - единственный корень уравнения f(x)=0, \; a\le t\le b.

(Мы не рассматриваем случай, когда корней на отрезке [a,b] несколько, то есть более одного. В качестве \eps можно взять и другое достаточно малое положительное число, например, 0,001.)

Поделим отрезок [a,b] пополам. Получим точку c= \frac {a+b}{2}, \; a<c<b и два отрезка [a,c], \; [c,b].

* Если f(c)=0, то корень t найден (t=c).
* Если нет, то из двух полученных отрезков [a,c] и [c,b] надо выбрать один [a_1;b_1] такой, что f(a_1)\cdot f(b_1)<0, то есть
  + [a_1;b_1] = [a,c], если f(a)\cdot f(c)<0 или
  + [a_1;b_1] = [c,b], если f(c)\cdot f(b)<0.

Новый отрезок [a_1;b_1] делим пополам. Получаем середину этого отрезка c_1=\frac {a_1+b_1}{2} и так далее.

Для того, чтобы найти приближённое значение корня с точностью до  \eps >0, необходимо остановить процесс половинного деления на таком шаге n, на котором |b_n-c_n|<\eps и вычислить x=\frac {a_n+b_n}{2}.

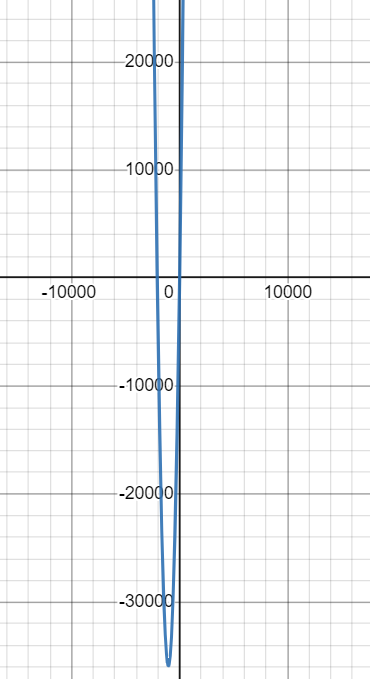
**Решение:**

Мое уравнение

****

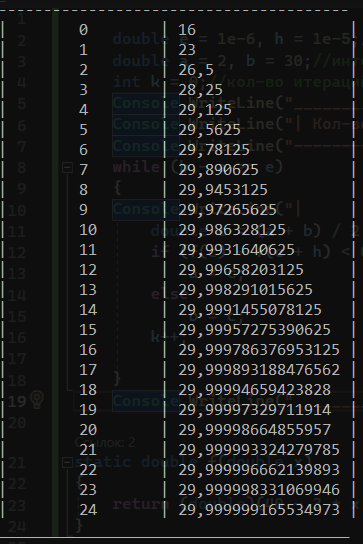
**Графики:**

Как видно из уравнения и графика ниже это парабола, которая на отрезке 2; 30 является прямой, соответственно из этого видно что минимум на отрезке будет в точке х=2, максимум х=30

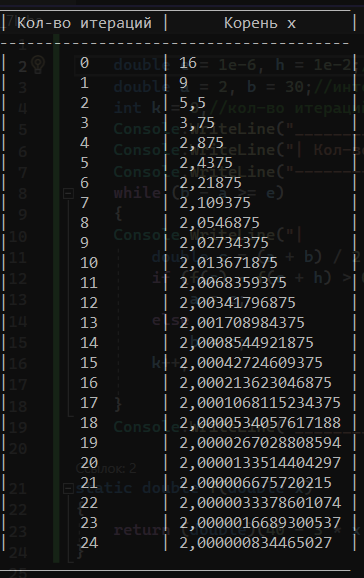
****

**Скриншоты:**

Максимум

****

Минимум



**Код:**

double e = 1e-6, h = 1e-2;//погрешность вычисления

double a = 2, b = 30;//интервал [2, 30]

int k = 0;//кол-во итераций

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine("| Кол-во итераций | Корень x |");

Console.WriteLine("---------------------------------------");

while (b - a >= e)

{

Console.WriteLine("| {0, -9}| {1, -18} |", k, (a + b) / 2);//выводим таблицу

double c = (a + b) / 2;

if (f(c) - f(c + h) > 0)//> min, < max

a = c;

else

b = c;

k++;

}

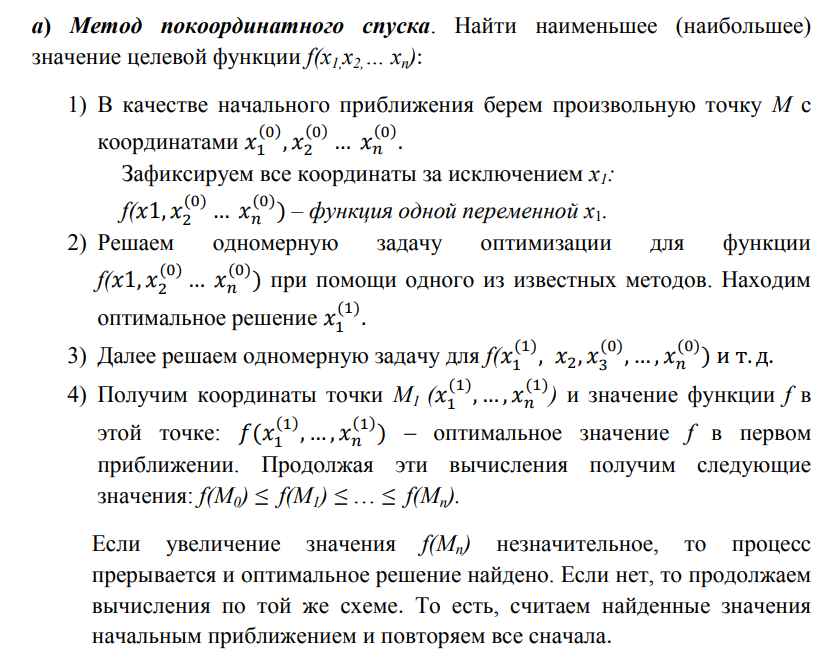
Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

static double f(double x)

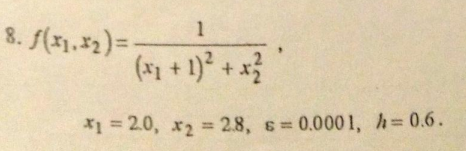
{

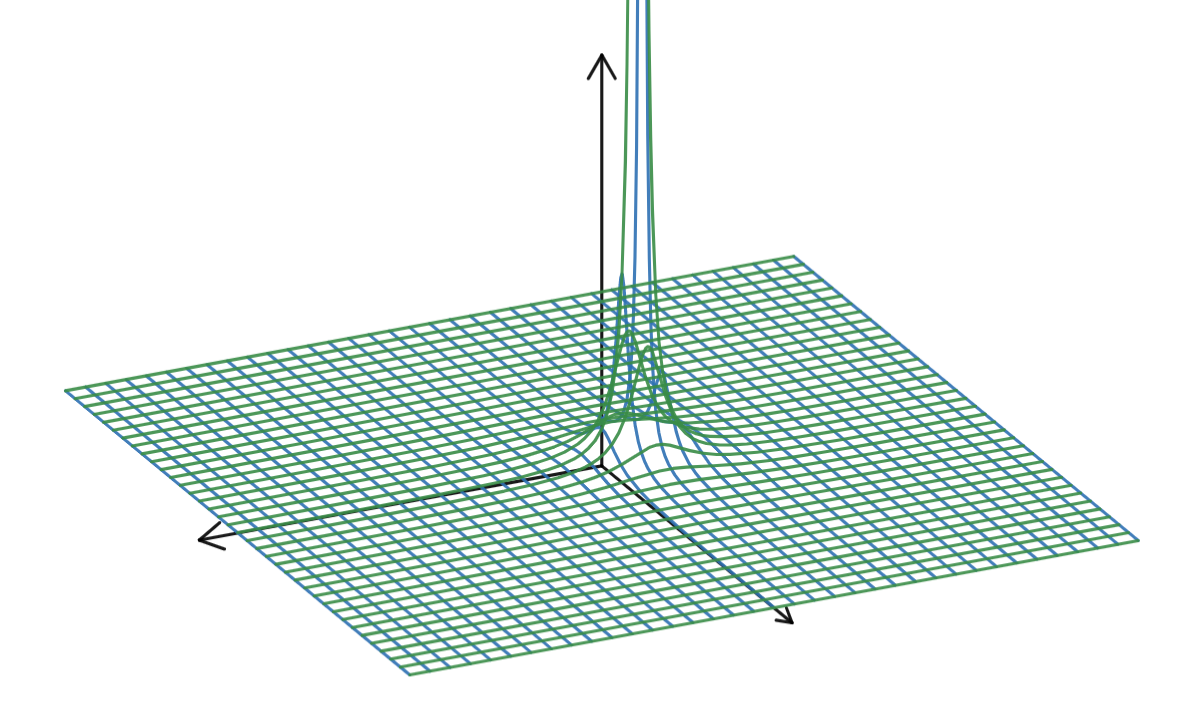
return (double)(40 - 3 \* x / 4 + x \* x / 30 + 70 \* x);

}

****

**Решение:**

****

**График**:

**Скриншот:**



**Код:**

double d1(double x)

{

double e = 1e-4, h = 0.6;//погрешность вычисления

double a = -50, b = 50;//интервал

while (b - a >= e)

{

double c = (a + b) / 2;

if (f(x, c) - f(x, c + h) < 0)//> min, < max

a = c;

else

b = c;

}

if (a < b) { return a; }

else { return b; }

}

double d2(double x)

{

double e = 1e-4, h = 0.6;//погрешность вычисления

double a = -50, b = 50;//интервал

while (b - a >= e)

{

double c = (a + b) / 2;

if (f(c, x) - f(c + h, x) < 0)//> min, < max

a = c;

else

b = c;

}

if (a < b) { return a; }

else { return b; }

}

double f(double x1, double x2)

{

return 1 / ((x1 + 1) \* (x1 + 1) + x2 \* x2);

}

double eps = 1e-4;

double x1 = 2, x2 = 2.8, xt1 = 0, xt2 = 0;

do

{

xt1 = x1;

xt2 = x2;

x2 = d1(x1);

x1 = d2(x2);

} while (Math.Abs(xt1 - x1) > eps && Math.Abs(xt2 - x2) > eps);

Console.WriteLine("x1= " + x1 + " " + "x2= " + x2);

**Вывод:** Оценка метода: Преимущество: не надо вычислять производную. Недостатки: 1. Находит только локальный экстремум; 2. Функция должна быть непрерывной в области поиска; 3. Существенно зависит от выбора начальной точки и направления движения; 4. Метод не работает в случае сильно вытянутых функций. В итоге можно скачать, что метод координатного спуска является простым в реализации методом оптимизации, главным недостатком метода является его ограниченная применимость