Министерство образования и науки РФ

ФГБОУ ВПО «Тамбовский государственный технический университет»

Кафедра «»

Лабораторная работа №6

по дисциплине «Вычислительная математика»

«Методы одномерной оптимизации САПР»

Вариант 15

Выполнил: студент гр. qwinmen.

Проверил:.

Тамбов, 20

**Задача:**

Составить блок-схему алгоритма и реализовать его в программе на ЭВМ для следующих методов одномерной оптимизации:

1. Метод дихотомии
2. Метод «золотого сечения»
3. Метод Фибоначчи

**Исходные данные:**

Нелинейная функция — , отрезок, содержащий корень, — [1;3], заданная точность — .

**Стратегия поиска.**

Метод дихотомии относится к последовательным стратегиям и позволяет исключить из дальнейшего рассмотрения на каждой итерации ровно половину текущего интервала неопределенности. Задается начальный интервал неопределенности и требуемая точность.

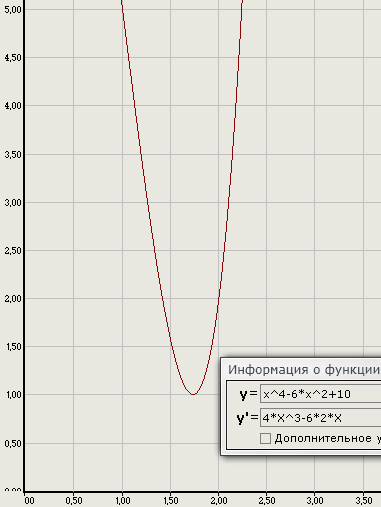
Алгоритм основан на анализе значений функции в трех точках, равномерно распределенных на текущем интервале (делящих его на четыре равные части). Поиск заканчивается, когда длина текущего интервала неопределенности оказывается меньше установленной величины.

**Принцип работы метода половинного деления:**

Разделим исходный отрезок пополам: . Проверяя знаки выясним в каком из отрезков или содержится корень:

Выбранный отрезок принимаем за и повторяем до тех пор, пока получаемый отрезок не сожмется до заданной степени точности.

График функции:

Решение аналитически:

Вычислить среднюю точку

длину интервала

и значение функции в средней точке

Вычислить точки

а также значения функции в этих точках:

Cравнить значения

Т.к. то из дальнейшего рассмотрения исключаем интервал

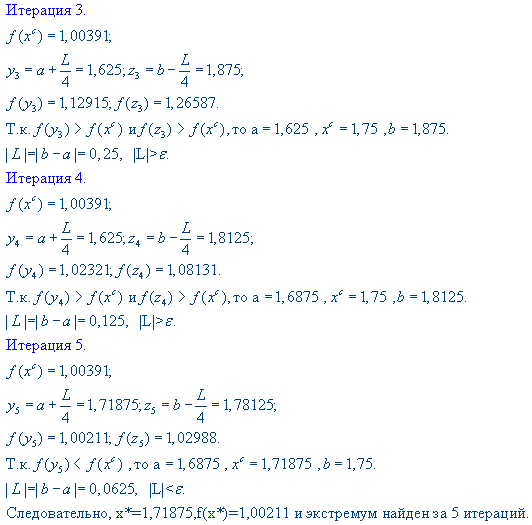
и следовательно решение продолжается, переходим к итерации 2.

Итерация 2:

[1; 2]

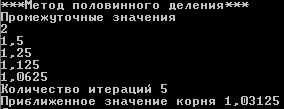
Т.к. , то исключается интервал [a, ), берем а==1.5, b=b=2, а средней точкой нового интервала станет точка z: =z=1.75

и следовательно решение продолжается, переходим к итерации 3.

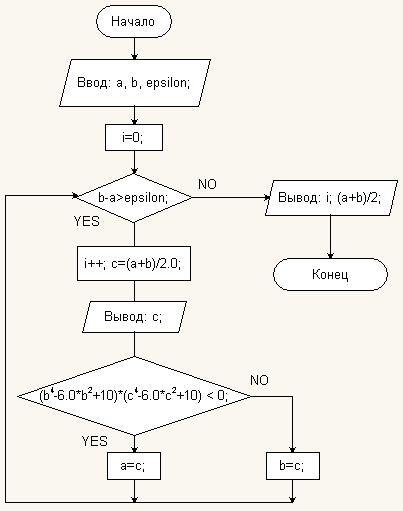


Ввиду медленной сходимости этот метод редко используется для нахождения значения корня, обычно его применяют для локализации корня с дальнейшим уточнением значения корня каким-либо другим методом.

Решение на ЭВМ:



Блок схема программы:



Исходный код программы:

class Дихотомия

{

static double f(double x)

{

return (Math.Pow(x,4.0)-6.0\*Math.Pow(x,2.0)+10.0);

}

public static void halfDivision(double a, double b, double epsilon)

{

int i = 0;

double c;

Console.WriteLine("\*\*\*Метод половинного деления\*\*\*\nПромежуточные значения");

while (b - a > epsilon)

{

i++;

c = (a + b) / 2.0;

Console.WriteLine(c);

if (f(b) \* f(c) < 0)

a = c;

else

b = c;

}

Console.WriteLine("Количество итераций {0}", i);

Console.WriteLine("Приближенное значение корня {0}", (a + b) / 2.0);

}

}

**Метод Золотого сечения.**

***Определение:***. Точка производит «золотое сечение» отрезка, если отношение длины всего отрезка к большей части равно отношению большей части к меньшей.

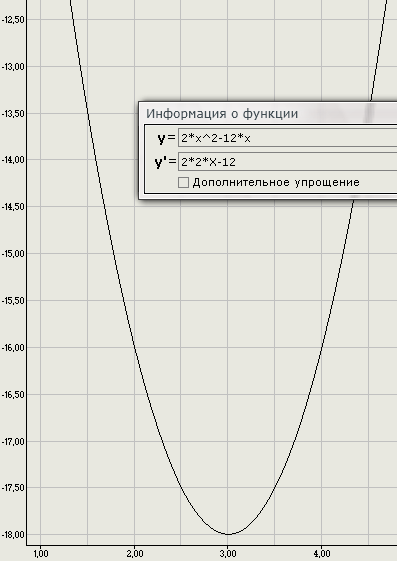
**Стратегия поиска.**

Метод относится к последовательным стратегиям. Задается начальный интервал неопределенности и требуемая точность. Алгоритм основан на анализе величин функции в двух точках. В качестве точек вычисления функции выбираются точки золотого сечения. Тогда с учетом свойств золотого сечения на каждой итерации, кроме первой, требуется только одно новое вычисление функции. Поиск заканчивается, когда длина текущего интервала неопределенности оказывается меньше установленной величины.

**Исходные данные:**

Определить минимум функции , заданной на отрезке — [0;10], при заданной точности — .

График функции:

Решение аналитически:

Итерация 1:

Вычислить точки

=

Вычислить значения

Cравнить значения :

т.к. , то исключается интервал взять

Вычислить

Т.к. то переходим к следующей итерации.

Итерация 2:

Т.к. то переходим к следующей итерации.

Итерация 3:

Т.к. то переходим к следующей итерации.

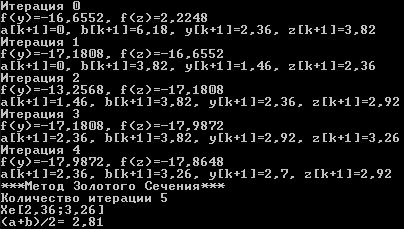
Итерация 4:

Т.к. то переходим к следующей итерации.

Итерация 5:

Т.к. , следовательно . N=6 итерации.

Решение на ЭВМ:



Исходный код программы:

class ЗолотоеСечение

{

static double f(double x)

{return (2.0 \* Math.Pow(x, 2) - 12.0 \* x);}

static double Delta(double a,double b)

{return Math.Abs(a - b);}

static double A\_B\_Z(double a,double b,double y\_z)

{return a + b - y\_z;}

static public void GoldMain(double a0, double b0, double epsilon)

{//[a;b]

if (epsilon < 0) Console.WriteLine("Точность не ниже нуля!!");

//step 2

int k = 0;

double f\_y, f\_z;

double[] y=new double[10],a=new double[10],z=new double[10], b=new double[10];

a[k] = a0;b[k] = b0;

y[k] = a[k] + 0.382\*(b[k] - a[k]);

z[k] = A\_B\_Z(a[k], b[k], y[k]);

step4:

Console.WriteLine("Итерация {0}",k);

f\_y = f(y[k]);

f\_z = f(z[k]);

Console.WriteLine("f(y)={0}, f(z)={1}",f\_y,f\_z);

if (f\_y<=f\_z)

{

a[k + 1] = a[k];

b[k + 1] = z[k];

y[k + 1] = A\_B\_Z(a[k + 1], b[k + 1], y[k]);

z[k + 1] = y[k];

Console.WriteLine("a[k+1]={0}, b[k+1]={1}, y[k+1]={2}, z[k+1]={3}",a[k+1],b[k+1],y[k+1],z[k+1]);

}

else

{

a[k + 1] = y[k];

b[k + 1] = b[k];

y[k + 1] = z[k];

z[k + 1] = A\_B\_Z(a[k + 1], b[k + 1], z[k]);

Console.WriteLine("a[k+1]={0}, b[k+1]={1}, y[k+1]={2}, z[k+1]={3}", a[k + 1], b[k + 1], y[k + 1], z[k + 1]);

}

if (Delta(a[k+1],b[k+1])<=epsilon)

{

Console.WriteLine("\*\*\*Метод Золотого Сечения\*\*\*\nКоличество итерации {0}",k+1);

Console.WriteLine("Xe[{0};{1}]",a[k+1],b[k+1]);

Console.WriteLine("(a+b)/2= {0}",(a[k+1]+b[k+1])/2.0);

}

else

{

k = k + 1;

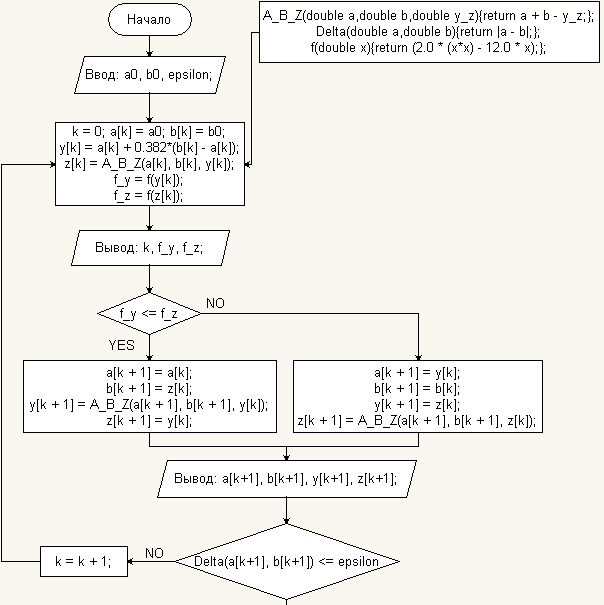
goto step4;

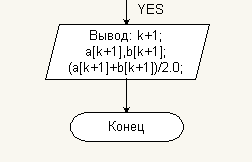
}

}

}

Блок схема программы:





**Метод Фибоначчи.**

В методе Фибоначчи реализована стратегия, обеспечивающая максимальное гарантированное сокращение интервала неопределенности при заданном количестве вычислений функции. Эта стратегия опирается на числа Фибоначчи.

***Определение:***

Числа Фибоначчи определяются следующим образом:

Последовательность чисел Фибоначчи имеет вид 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233…

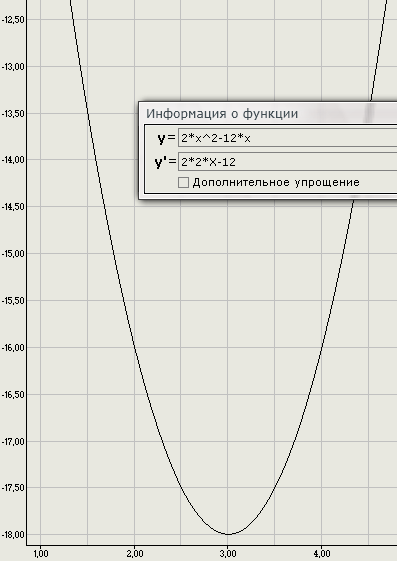
**Стратегия поиска:**

Метод относится к последовательным стратегиям. Задается начальный интервал неопределенности и количество N вычислений функции. Алгоритм основан на анализе величин функции в двух точках. Точки вычисления функции находятся с использованием последовательности из N+1 чисел Фибоначчи. Как и в методе золотого сечения, на первой итерации требуется два вычисления функции, а на каждой последующей итерации, требуется только одно новое вычисление функции. Поиск заканчивается, когда длина текущего интервала неопределенности оказывается меньше установленной величины.

**Исходные данные:**

Определить методом Фибоначчи минимум функции заданной на отрезке [0; 10], при допустимой длине конечного интервала l=1, e=0.01;

График функции:

Решение аналитически: 

Найдем количество N вычислений функции как наименьшее целое число, при котором удовлетворяется условие , и числа Фибоначчи

Зададим , поэтому N=6.

Найдем числа Фибоначчи:

Итерация 1:

Вычислить значения =;

Вычислить :

Сравнить :

Т.к. , то берем ; ; ;

Проверить условие окончания и в случае необходимости сделать заключительное N-е вычисление функции для получения решения:

Cледовательно переходим к следующей итерации.

Итерация 2:

Т.к. , то

Проверяем условие окончания:

Итерация 3:

Т.к. то

Проверяем условие окончания:

Итерация 4:

Т.к. то

Проверяем условие окончания:

Т.к. выполнимо равенство с k, то всегда , т.е. отсутствует точка нового вычисления функции. В этом случае полагают

Итерация 5:

В точках вычисляют значения функции и находят границы конечного интервала неопределенности:

Т.к. то

Поэтому

Результат на ЭВМ:



Исходный код программы:

class ЧислаФибоначчи

{

static int[] ЧислаФибо=new int[10];

public static int Fibo(double n)

{//n=10;

int a = 1; int b = 1; int temp = 1; int i;

ЧислаФибо[0] = ЧислаФибо[1] = 1;

for (i = 1; i < n; i++ )

{

temp = a + b;

a = b;

b = temp;

ЧислаФибо[i+1] = temp;

if(temp>=n) break;

}

Console.WriteLine("F={0} > n={1} ==>N={2}",temp,n,i+1);

return i+1;

}

static double f(double x)

{ return (2.0 \* Math.Pow(x, 2) - 12.0 \* x); }

static double A\_F\_F(double a, double b, double F\_числит,double F\_знаменат)

{ return a + (F\_числит/F\_знаменат)\*(b-a); }

public static void Фибоначи(double a0,double b0,double epsilon, double l)

{

double n = (b0 - a0)/l;//10

int N = Fibo(n), k = 0;

double f\_y, f\_z;

double[] y = new double[10], a = new double[10], z = new double[10], b = new double[10];

a[k] = a0;b[k] = b0;

y[k] = A\_F\_F(a[k], b[k], ЧислаФибо[N - 2], ЧислаФибо[N]);

z[k] = A\_F\_F(a[k], b[k], ЧислаФибо[N - 1], ЧислаФибо[N]);

step5:

f\_y = f(y[k]);

f\_z = f(z[k]);

if (f\_y<=f\_z)

{

a[k + 1] = a[k];

b[k + 1] = z[k];

z[k + 1] = y[k];

y[k + 1] = A\_F\_F(a[k + 1], b[k + 1], ЧислаФибо[N - k - 3], ЧислаФибо[N - k - 1]);

}

else

{

a[k + 1] = y[k];

b[k + 1] = b[k];

y[k + 1] = z[k];

z[k + 1] = A\_F\_F(a[k + 1], b[k + 1], ЧислаФибо[N - k - 2], ЧислаФибо[N - k - 1]);

}

if (k!=N-3)

{

k = k + 1;

goto step5;

}

else

{

y[N - 2] = z[N - 2] = (a[N - 2] + b[N - 2])/2;

y[N - 1] = y[N - 2] = z[N - 2];

z[N - 1] = y[N - 1] + epsilon;

if (f(y[N-1])<=f(z[N-1]))

{

a[N - 1] = a[N - 2];

b[N - 1] = z[N - 1];

Console.WriteLine("\*\*\*Метод Фибоначчи\*\*\*");

Console.WriteLine("Xe[{0:#.####};{1:#.####}]", a[N - 1], b[N - 1]);

Console.WriteLine("({0:#.####}+{1:#.####})/2={2:#.####}", a[N - 1], b[N - 1], (a[N - 1] + b[N - 1]) / 2.0);

}

else

{

a[N - 1] = y[N - 1];

b[N - 1] = b[N - 2];

Console.WriteLine("\*\*\*Метод Фибоначчи\*\*\*");

Console.WriteLine("Xe[{0:#.####};{1:#.####}]", a[N - 1], b[N - 1]);

Console.WriteLine("({0:#.####}+{1:#.####})/2={2:#.####}", a[N - 1], b[N - 1], (a[N - 1] + b[N - 1]) / 2.0);

}

}

}

}

Блок схема программы

