Санкт-Петербургский государственный университет

**Курсовая работа**

по дисциплине «Численные методы»

на тему: «Методы уточнения корней»

Студент: Марусина Анастасия Петровна

Группа: 209

Преподаватель: Перегудин Сергей Иванович

Метод простых итераций

Приближенное решение уравнения *f(x)=0* методом простых итераций

Теоретическое обоснование метода

Теорема: (Достаточное условие сходимости).

Пусть функция *Ф(х)* определена и непрерывно дифференцируема на отрезке [a,b], причем на отрезке [a,b] функция принимает значения из [a,b], т.е. *Ф(х)∈[a,b]*

Тогда ∃q такое, что |*Ф’(x)*|≤q<1, и тогда, исходя из любого начального приближения *xo∈[a,b]* итерационная последовательность *xn+1 = Ф(xn)* сходится к точному решению уравнения:



*( ξ)* = 0; *ξ∈[a,b],*



причем, если *Ф(х)* монотонно возрастающая, то можно оценить точность определения корня:

если *Ф’(x)>0,* то



если *Ф’(x)<0,* то

где ξ - точный корень,



*xn -* корень, найденный в процессе итерации.

Построение функции Ф(х).

Для сходимости при любом начальном приближении *x0*∈[a,b] функция Ф(х) выбирается такой, что:

* + Ф(х) дифференцируема на [a,b] .
  + Ф(х) ∈ [a,b] при *x*∈[a,b].

∃q такое, что | Ф`(x)| ≤ q < 1 на отрезке [a,b]

Текст программы

#include "stdafx.h"

#include <conio.h>

#include <math.h>

float f (float);

float g (float);

float c;

int main(int argc, char\* argv[])

{

float x0,xk,a,b,epsilon;

printf("Нахождение корня функции f(x) = 0 с начальным приближением x0 \nметодом простых итераций\n");

printf("Вычисления производятся для функции f(x)=ax^3+bx^2+cx+d, при \na=0.1697\nb=-0.5693\nc=-1.6\nd=3.73\n");

printf("Левая граница отрезка: ");

scanf("%f",&a);

printf("Правая граница отрезка: ");

scanf("%f",&b);

printf("Начальное приближение: ");

scanf("%f",&x0);

printf("Точность вычислений: ");

scanf("%f",&epsilon);

c=0.1f;

do

{

xk=g(x0);

if (fabs(xk-x0)<epsilon) break;

else x0=xk;

}

while (fabs(a-x0)>epsilon && fabs(b-x0)>epsilon);

printf("Значение корня: %f\n",xk);

getch();

return 0;

}

float f (float x)

{

const float a=0.1697f;

const float b=-0.5693f;

const float c=-1.6f;

const float d=3.73f;

return (a\*x\*x\*x + b\*x\*x + c\*x+d);

}

float g (float x)

{

return c\*f(x)+x;

}

Метод Дихотомии

Теоретическое обоснование метода

## Теорема Больцано-Коши

Если функция *f(x)* на концах принимает значения разных знаков: *f(a)\*f(b)<0,* то тогда ∃ такое значение *x=c,* что *f(c)=0; c∈[a,b],* причем корень будет единственен, если производная на рассматриваемом интервале не меняет знака.

Алгоритм решения

* + Получаем границы отрезка  и требуемую точность вычислений
  + (\*) Считаем середину отрезка: 
  + Если *f(c)\*f(a) < 0*, то b:=a
  + Если *f(c)\*f(a) >= 0*, то b:=c
  + Если *f(c) = 0,* то проверяем знаки на концах отрезка [c-eps/2, c+eps/2]. Если функция принимает значения разных знаков, то корень найден. Если нет, то произошла ошибка, и программа аварийно выходит.
  + Перейти к шагу (\*), если длина нового отрезка  , или значение функции в точке «c» - больше заданной точности.

Вывести найденное значение корня

Текст программы

#include "stdafx.h"

#include <math.h>

#include <conio.h>

#define FALSE 0

#define TRUE 1

// Tested with params

// [4;5]

// 0.00001

float f (float);

int main(int argc, char\* argv[])

{

float a,b,c,epsilon;

char blnError = FALSE;

printf("Нахождение корня функции f(x) = 0 на отрезке [n1,n2] \nметодом деления пополам (бисекции)\n");

printf("Вычисления производятся для функции f(x)=tg(ax)-bx, \nпри a=0.9464 и b=1.3825\n");

printf("Введите границы отрезка для нахождения корня\n");

printf("Левая граница отрезка: "); scanf("%f",&a);

printf("Правая граница отрезка: "); scanf("%f",&b);

printf("Точность вычислений: "); scanf("%f",&epsilon);

do

{

c=(a+b)/2;

((f(c)\*f(a)<0)?b:a)=c;

if (f(c)==0)

{

printf ("Значение функции в точке %f равно нулю\n",c);

blnError = (f(c-epsilon/2)\*f(c+epsilon/2)<0)?FALSE:TRUE;

break;

}

}

while (fabs(f(c))>epsilon || fabs(b-a)>epsilon);

if (blnError)

{

printf ("Однако, на концах отрезка [%f-%f,%f+%f] \nфункция не принимает значения разных знаков\n",c,epsilon,c,epsilon);

}

else

{

printf("Значение корня: %f\n",c);

}

getch();

return 0;

}

float f (float x)

{

const float a = 0.9464; const float b = 1.3825;

return tan(a\*x)-b\*x;

};

Метод касательных

Теоретическое обоснование метода

Метод касательных для решения уравнений вида (1) для произвольной функции f(x) состоит в построении итерационной последовательности:



сходящейся при n→∞ к корню уравнения.

Метод эффективно применим, если подобрано хорошее начальное приближение xo и гарантирует высокую скорость сходимости.

Теорема: (Достаточные условия сходимости).

Пусть функция f(x) определена и дважды непрерывно дифференцируема, причем производные f’(x) и f’’(x) сохраняют знаки на отрезке [a,b] (т.е. на этом отрезке есть корень),

тогда исходя из начального приближения xo , удовлетворяющего условию

f(xo)\*f’’(xo)>0 итерационный процесс (\*) сходится к единственному на [a,b] корню ξ : f(ξ)=0, ξ ∈ [a,b]

Текст программы

#include "stdafx.h"

#include <conio.h>

#include <math.h>

const float a=2.113f;

const float b=-6.44f;

const float c=-3.19f;

const float d=15.13f;

const float m=2.41064f; const float M=20.0828f;

float f (float);

float df (float);

int main(int argc, char\* argv[])

{

float x0,xk,a,b,epsilon;

printf("Нахождение корня функции f(x) = 0 с начальным приближением x0 \nметодом Ньютона\n");

printf("Вычисления производятся для функции f(x)=ax^3+bx^2+cx+d, при \na=2.113\nb=-6.44\nc=-3.19\nd=15.13\n");

printf("Левая граница отрезка: ");

scanf("%f",&a);

printf("Правая граница отрезка: ");

scanf("%f",&b);

printf("Начальное приближение: ");

scanf("%f",&x0);

printf("Точность вычислений: ");

scanf("%f",&epsilon);

xk=x0;

do

{

x0=xk;

xk=x0-(f(x0)/df(x0));

}

while (fabs(xk-x0)>sqrt(fabs(2.f\*epsilon\*m/M)));

printf("Значение корня: %f\n",x0);

getch();

return 0;

}

float f (float x)

{

return (a\*x\*x\*x + b\*x\*x + c\*x + d);

}

float df (float x)

{

return (3\*a\*x\*x + 2\*b\*x + c);

}

Метод хорд

 Если http://www.pm298.ru/Math/f885.JPG, http://www.pm298.ru/Math/f886.JPG - приближенные значения корня уравнения *f(x) = 0*, а http://www.pm298.ru/Math/f900.JPG то последующие приближения находят по формуле

http://www.pm298.ru/Math/f901.JPG

     Методом хорд называют также метод, при котором один из концов отрезка http://www.pm298.ru/Math/f897.JPG закреплен, т. е. вычисление приближения корня уравнения *f(x) = 0* производят по формулам:

http://www.pm298.ru/Math/f902.JPG

либо

http://www.pm298.ru/Math/f903.JPG

     При этом предполагается, что корень уравнения находится на отрезке http://www.pm298.ru/Math/f897.JPG, а *f''(x)* сохраняет знак на http://www.pm298.ru/Math/f897.JPG

Текст программы

#include <stdio.h>

#include <math.h>

// this function returns value of f(x)

double f(double x)

{

return 2 \* x\*x\*x - x\*x - .46;

}

double Solve(double a, double b, double epsilon)

{

double u, v, x, y;

u = f(a); v = f(b);

do

{

// x - new border

x = (a\*v - b\*u) / (v - u);

y = f(x);

// determine whether x is left or right border

// f(a)\*f(b)<0 condition must be true

if (y\*u<0)

{

b = x;

v = y;

}

else

{

a = x;

u = y;

}

printf("x: %lf\n", x);

} while (b - a>epsilon);

return x;

}

int main(void)

{

printf("2\*x\*x\*x - x\*x - .46 = 0, [a; b]=[0.5; 1.0]");

printf("\nnx0=%f", Solve(0.5, 1.0, 1e-6));

}

Комбинированный метод хорд и касательных

Методы хорд и касательных дают приближения корня с разных сторон. Поэтому их часто применяют в сочетании друг с другом, тогда уточнение корня происходит быстрее.

Пусть дано уравнение *f(x) = 0*, корень отделен на отрезке *[a, b]*.

Рассмотрим случай, когда *f ‘(x) f ’’(x)>0*

В этом случае метод хорд дает приближенное значение корня с недостатком (конец *b* неподвижен), а метод касательных – с избытком (за начальное приближение берем точку *b*).

Тогда вычисления следует проводить по формулам:

http://dit.isuct.ru/ivt/sitanov/Literatura/M501/Pages/Glava2_5.files/image003.gif

http://dit.isuct.ru/ivt/sitanov/Literatura/M501/Pages/Glava2_5.files/image005.gif

Теперь корень *ξ* заключен в интервале *[a1, b1]*. Применяя к этому отрезку комбинированный метод, получим:

http://dit.isuct.ru/ivt/sitanov/Literatura/M501/Pages/Glava2_5.files/image007.gif

http://dit.isuct.ru/ivt/sitanov/Literatura/M501/Pages/Glava2_5.files/image009.gif

и т.д.

|  |  |
| --- | --- |
| http://dit.isuct.ru/ivt/sitanov/Literatura/M501/Pages/Glava2_5.files/image011.gif  http://dit.isuct.ru/ivt/sitanov/Literatura/M501/Pages/Glava2_5.files/image013.gif | (2.6) |

Если же *f ‘(x) f ’’(x)<0*  то, рассуждая аналогично, получим следующие формулы для уточнения корня уравнения

http://dit.isuct.ru/ivt/sitanov/Literatura/M501/Pages/Glava2_5.files/image015.gif

http://dit.isuct.ru/ivt/sitanov/Literatura/M501/Pages/Glava2_5.files/image017.gif

Вычислительный процесс прекращается, как только выполнится условие:

http://dit.isuct.ru/ivt/sitanov/Literatura/M501/Pages/Glava2_5.files/image019.gif

Текст программы:

using namespace std;

static const double eps = 0.0001;

double polinom(double x){// само уравнение

return (x-sin(x)-0.25);

}

class Equation

{

public:

Equation()

{

}

double derivative(double x)

{

double h = eps;

double temp1;

double temp2 = (polinom(x+h)-polinom(x-h))/(2\*h);

h /= 2;

do

{

temp1 = temp2;

temp2 = (polinom(x+h)-polinom(x-h))/(2\*h);

h /= 2;

}

while (abs(temp1-temp2)>=eps);

return temp2;

}

double secondDerivative(double x)

{

double h = eps;

double temp1;

double temp2 = (polinom(x+h)-2\*polinom(x)+polinom(x-h))/(2\*h);

h /= 2;

do

{

temp1 = temp2;

temp2 = (polinom(x+h)-2\*polinom(x)+polinom(x-h))/(2\*h);

h /= 2;

} while (abs(temp1-temp2)>=eps);

return temp1;

}

double tangentChordsMethod(double a, double b)

{

int n = 0;

while (abs(a-b)>=eps)

{

if (derivative(a) \* secondDerivative(a) > 0)

{

a -= polinom(a)\*(b-a)/(polinom(b)-polinom(a));

b -= polinom(b)/derivative(b);

}

else

{

b -= polinom(b)\*(b - a)/(polinom(b)-polinom(a));

a -= polinom(a)/derivative(a);

}

n++;

}

cout << "Count of iterations: n = " << n ;

return (a + b) / 2;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

double a=1;

double b=2;

cout <<endl<< "Tangents'&Chords' Method" << endl<<"x\* = " <<tangentChordsMethod(a,b) << endl;

system ("pause");

return 0;

}