**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по практическому заданию №4**

**по дисциплине «Вычислительная математика»**

Тема: **Решение нелинейных уравнений. Метод Ньютона**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Виноградов К.А. |
| Преподаватель |  | Лисс А.Р. |

Санкт-Петербург

2017

**Цель работы.**

Найти корень уравнения  методом Ньютона с заданной точностью Eps, исследовать обусловленность метода (чувствительность к ошибкам в исходных данных) и скорость сходимости метода.

**Основные теоретические положения.**

Пусть задана непрерывная функция  вещественного аргумента x и требуется численным методом решить уравнение , т.е. найти приближение x\* к вещественному корню этого уравнения. Если уравнение имеет несколько вещественных корней, то сначала производят их отделение (изоляцию), а затем уточняют положение отдельного корня. Считается, что отделение корня произведено, если выделен такой интервал [a0, b0] области определения функции , на концах которого значения функции (a0) и (b0) имеют разные знаки и внутри которого имеется ровно один корень уравнения. В случае, когда известно хорошее начальное приближение решения уравнения , эффективным методом повышения точности является метод Ньютона. Он состоит в построении итерационной последовательности  сходящейся к корню уравнения.

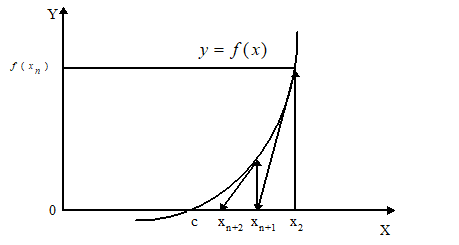


Рисунок 1 – Метод Ньютона

Геометрическая интерпретация метода Ньютона представлена на рис. 1. Если через точку с координатами  провести касательную, то абсцисса точки пересечения этой касательной с осью Ох будет очередным приближением корня уравнения .

Для оценки погрешности n-го приближения корня предлагается пользоваться неравенством

 (1)

где -наибольшее значение модуля второй производной  на отрезке [a,b]; -наименьшее значение модуля первой производной  на отрезке [a,b].

Таким образом из (1) следует, что если , то .

Это означает, что при хорошем начальном приближении корня после каждой итерации число верных десятичных знаков в очередном приближении удваивается, т.е. процесс сходится очень быстро (имеет место квадратичная сходимость). Из указанного следует, что при необходимости нахождения корня с точностью ε итерационный процесс можно прекращать, когда

  (2)

Рассмотрим один шаг итераций. Если на (n-1)-м шаге очередное приближение неудовлетворяет условию окончания процесса, то вычисляются величины ,  и следующие приближение корня . При выполнении условия (2) величина  принимается за приближенное значение корня с, вычисленное с точностью ε.

**Выполнение работы.**

Дана функция 

Графически определим отрезок, на котором находится корень уравнения. График представлен на рис. 2.

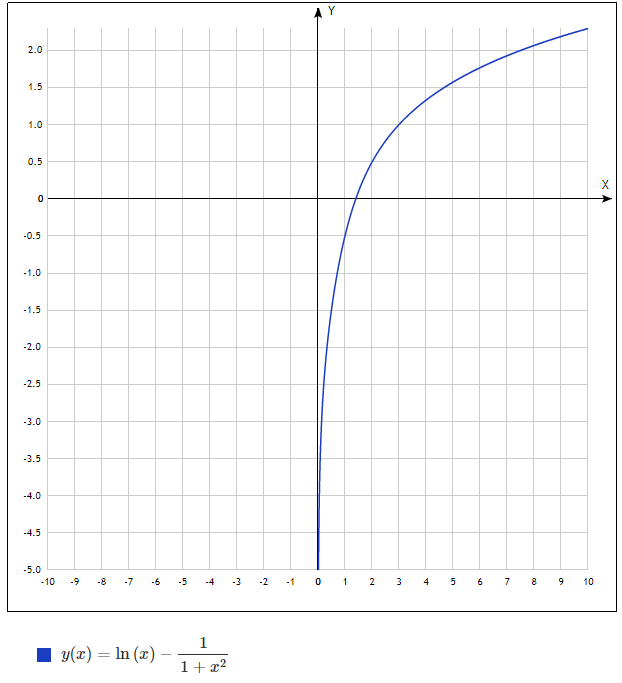


Рисунок 2 – График уравнения

Нужный отрезок ­­– [1.0; 2.0].

Составим подпрограммы вычисления функции и . Код подпрограммы находится в приложении А.

Составим головную программу содержащую обращение к подпрограммам , , NEWTON, Round и индикации результатов. Код программы находится в приложении А.

Выберем начальное приближение  из [1,2] так, чтобы выполнялось условие . Пусть .

Для нахождения числа абсолютной обусловленности возьмём производную от :

 (3)

Соответственно, число обусловленности можно легко получить из (3), как модуль обратной величины:

 (4)

Исследуем чувствительность метода к ошибкам в исходных данных. Для этого зафиксируем значение погрешности результата Eps = 0.0001. А значение погрешности исходных данных будем изменять. Кроме того, зафиксируем количество итераций, которое понадобилось для вычисления корня. Число обусловленности будем вычислять по формуле (4). Результаты вычислений представлены в табл. 1.

Таблица 1 – Вычисления при постоянном Eps и переменном Delta

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Eps | Delta | x\* | k |  |
| 0.00001 | 0.1 | 1.400000 | 1 | 0.967247 |
| 0.00001 | 0.01 | 1.404352 | 2 | 0.970859 |
| 0.00001 | 0.001 | 1.401772 | 2 | 0.968717 |
| 0.00001 | 0.0001 | 1.401277 | 2 | 0.968307 |
| 0.00001 | 0.00001 | 1.401318 | 2 | 0.968340 |
| 0.00001 | 0.000001 | 1.401321 | 3 | 0.968341 |

Для расчёта скорости сходимости метода воспользуемся формулой:

, (5)

где *С*(0, 1), *k* .

Возьмём  из всех опытов в табл. 1, начиная со второго и полагая, **, оценим значение *p*. Результаты представлены в табл. 2.

Таблица 2 – Результаты вычисления скорости сходимости метода Ньютона

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0,004352 | 0,001772 | 0,0000189399 |
| 0,002580 | 0,003075 | 0,0000066564 |
| 0,000495 | 0,000454 | 0,0000002450 |
| 0,000041 | 0,000044 | 0,0000000017 |

Из этого следует, что p ~ 2 при . Значит скорость сходимости квадратичная.

**Выводы.**

В ходе выполненной работы был вычислен корень данного уравнения методом Ньютона с заданным значением точности. Была исследована скорость сходимости данного метода по табл. 2 и был получен квадратичный результат. Кроме того, исследовав чувствительность метода к ошибкам в исходных данных, можно сделать вывод, что данная задача обладает хорошей обусловленностью.

По сравнению с другими методами, методу Ньютона необходимо меньшее число итераций для нахождения корня, и при этом точность у него выше.

Приложение А

Код программы

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

extern double F(double);

extern double F1(double);

double Round (double X,double Delta);

double NEWTON (double X,double Eps,int &N);

double delta;

int main()

{

int k;

long int s;

float eps1,delta1;

double a,b,eps,x,x1;

double F(double);

printf("Insert eps:");

scanf("%f",&eps1);

eps = eps1;

printf("Insert x0:");

scanf("%f",&x1);

x = x1;

printf("Insert delta:");

scanf("%f",&delta1);

delta = delta1;

x = NEWTON(x,eps,k);

printf("x=%f k=%d\n",x,k);

return 0;

}

double F(double x)

{

extern double delta;

double s;

long int S;

s = log(x)-1/(1+x\*x);

if( s/delta < 0 )

S = s/delta - .5;

else

S = s/delta + .5;

s = S\*delta;

s = Round( s,delta );

return(s);

}

double F1(double x)

{

extern double delta;

double s;

long int S;

s = 2\*x/((x\*x+1)\*(x\*x+1))+1/x;

if( s/delta < 0 )

S = s/delta - .5;

else

S = s/delta + .5;

s = S\*delta;

s = Round( s,delta );

return(s);

}

double Round (double X,double Delta)

{

if (Delta<=1E-9) {puts("Wrong round accuracy\n");exit(1);}

if (X>0.0) return (Delta\*(long((X/Delta)+0.5)));

else return (Delta\*(long((X/Delta)-0.5)));

}

double NEWTON (double X,double Eps,int &N)

{

extern double F1 (double);

double Y,Y1,DX;

N=0;

do

{

Y = F(X);

if (Y==0.0) return (X);

Y1 = F1(X);

if (Y1==0.0) {puts("Derivative transformed into null\n");exit(1);}

DX=Y/Y1; X=X-DX; N++;

}

while (fabs(DX)>Eps);

return (X);

}