**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по практической работе №5**

**по дисциплине «Вычислительная математика»**

Тема: **Метод Ньютона**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 |  | Ларин А. |
| Преподаватель |  | Сучков А.И. |

Санкт-Петербург

2019

**Цель работы.**

Формирование практических навыков нахождения корней алгебраических и трансцендентных уравнений методом Ньютона.

**Основные теоретические положения.**

В случае, когда известно хорошее начальное приближение решения уравнения , эффективным методом повышения точности является метод Ньютона (касательных). Он состоит в построении итерационной последовательности , сходящейся к корню уравнения . Достаточные условия сходимости метода формулируются теоремой.

Теорема.

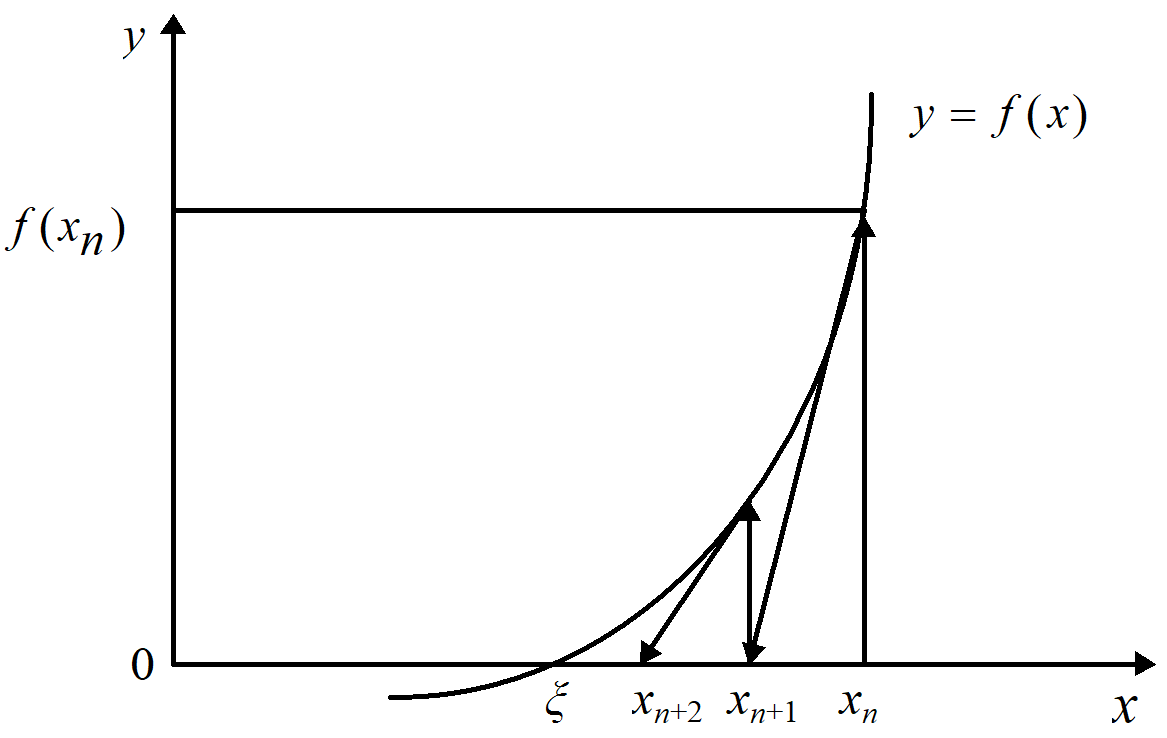
Пусть определена и дважды дифференцируема на причём , а производные сохраняют знак на отрезке . Тогда, исходя из начального приближения , удовлетворяющего неравенству , можно построить последовательность

сходящуюся к единственному на решению уравнения .

Метод Ньютона допускает простую геометрическую интерпретацию, представленную на рис. 1. Если через точку с координатами провести касательную, то абсцисса точки пересечения этой касательной с осью OX будет очередным приближением корня уравнения .

Для оценки погрешности -го приближения корня предлагается пользоваться неравенством:

где – наибольшее значение модуля второй производной на отрезке – наименьшее значение модуля первой производной на отрезке .

Рисунок 1 − Геометрическая интерпретация метода Ньютона

Таким образом, 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 квадратическая сходимость). Из указанного следует, что при необходимости нахождения корня с точностью ε итерационный процесс можно прекращать, когда

Рассмотрим один шаг итераций. Если на -м шаге очередное приближение не удовлетворяет условию окончания процесса, то вычисляются величины и следующие приближение корня . При выполнении условия остановки, описанного выше, величина принимается за приближенное значение корня , вычисленное с точностью .

**Постановка задачи.**

Используя подпрограммы-функции NEWTON и Round из файла methods.cpp (файл заголовков methods.h), найти корень уравнения с заданной точностью методом Ньютона, исследовать скорость сходимости и обусловленность метода. Порядок выполнения работы следующий:

1. Графически или аналитически отделить корень уравнения . Убедиться, что на найденном отрезке функция удовлетворяет условиям сходимости метода Ньютона.
2. Выбрать начальное приближение корня так, чтобы .
3. Оценить снизу величину , оценить сверху величину
4. По заданному выбрать значение для условия окончания итерационного процесса .
5. Составить подпрограммы-функции вычисления , предусмотрев округление их значений с заданной точностью .
6. Составить головную программу, вычисляющую корень уравнения и содержащую обращение к подпрограммам F, F1, Round, NEWTON и индикацию результатов.
7. Провести вычисления по программе. Исследовать скорость сходимости метода и его чувствительность к ошибкам в исходных данных, сравнить скорости сходимости методов Ньютона, бисекции и хорд.

**Выполнение работы.**

Проанализируем функцию :

Отделим графическим методом корни уравнения, т.е. найдем отрезки [Left, Right], на которых функция удовлетворяет условиям теоремы Больцано-Коши. По графику на рис.1 видно что корень принадлежит отрезку и функция на его концах принимает разные знаки.

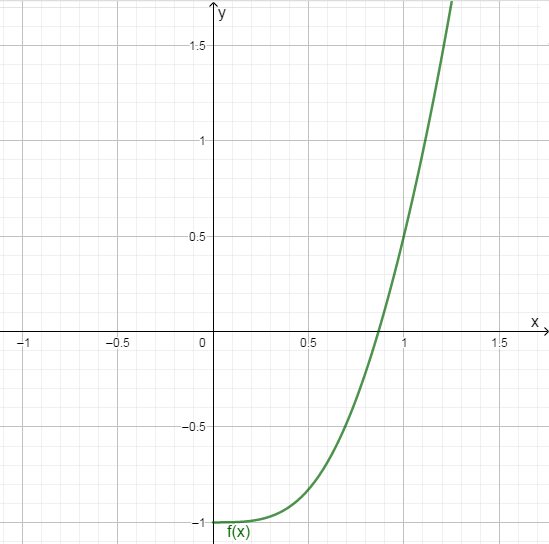


Рисунок 1 – Локализация корня функции

Проверим, что на выбранном отрезке функция удовлетворяет условиям сходимости метода ньютона:

* Функция дважды дифференцируема на
* Производные сохраняют знак

Возмем начальное приближение .

. Данное приближение соответствует условию теоремы.

Оценим величины :

,

*.*

Тогда

Проведем вычисление корня функций при помощи программы, приведенной в приложении А. Программа вычисляет корень уравнения методом Ньютона. На вход ей подаются следующие параметры: X – начальное приближение корня, eps – требуемая точность вычисления корня, delta – погрешность вычисления значений функции, a, b – отрезок , локализующий корень. В табл.1 приведены расчеты корня при различных значениях eps, и представлены значения количества итераций.

Таблица 1 – Расчет корня методом Ньютона с варьированием значения eps

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Значение eps | Значение delta | Значение a | Значение b | Значение | Значение |
| 0.1 | 0.00001 | 0.5 | 1 | 0.872045 | 2 |
| 0.01 | 0.00001 | 0.5 | 1 | 0.867107 | 3 |
| 0.001 | 0.00001 | 0.5 | 1 | 0.867107 | 3 |
| 0.0001 | 0.00001 | 0.5 | 1 | 0.867107 | 3 |
| 0.00001 | 0.00001 | 0.5 | 1 | 0.867086 | 4 |

Имея экспериментальное значение скорость сходимости метода видим, что скорость сходимости метода хорд выше линейной. Действительно, согласно теории порядок сходимости метода равен , т.е. квадратичный.

Теперь, имея приближение корня, примем . С помощью данного приближения вычислим , и оценим с его помощью чувствительность метода к ошибкам в исходных данных. При будем считать, что задача хорошо обусловлена – хор., иначе пл. – плохо. Результаты эксперимента занесены в табл.2. Теперь, имея количество итераций, необходимое для локализации корня, сравним мотод Ньютона с методом бисекции и методом хорд. Из проведенного ранее исследования известно, что порядок сходимости метода бисекции и метода хорд линейны

Таблица 2 – Обусловленность задачи при различных и

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Значение eps | Значение delta | Значение | Значение  *k* | Значение | Значение |
| 0.1 | 0.01 | 0.87243 | 2 | 0.005344 | хор. |
| 0.01 | 0.01 | 0.866566 | 3 | 0.00052 | хор. |
| 0.001 | 0.01 | 0.866566 | 3 | 0.00052 | хор. |
| 0.0001 | 0.01 | 0.866566 | 3 | 0.00052 | пл. |
| 0.00001 | 0.01 | 0.866566 | 3 | 0.00052 | пл. |
| 0.000001 | 0.01 | 0.866566 | 3 | 0.00052 | пл. |
| 0.1 | 0.001 | 0.872102 | 2 | 0.005016 | хор. |
| 0.01 | 0.001 | 0.867114 | 3 | 0.000028 | хор. |
| 0.001 | 0.001 | 0.867114 | 3 | 0.000028 | хор. |
| 0.0001 | 0.001 | 0.867114 | 3 | 0.000028 | хор. |
| 0.00001 | 0.001 | 0.867114 | 3 | 0.000028 | пл. |
| 0.000001 | 0.001 | 0.867114 | 3 | 0.000028 | пл. |
| 0.1 | 0.0001 | 0.872037 | 2 | 0.004951 | хор. |
| 0.01 | 0.0001 | 0.867108 | 3 | 0.000022 | хор. |
| 0.001 | 0.0001 | 0.867108 | 3 | 0.000022 | хор. |
| 0.0001 | 0.0001 | 0.867108 | 3 | 0.000022 | хор. |
| 0.00001 | 0.0001 | 0.867078 | 4 | 0.000008 | хор. |

В табл.3 приведено сравнение методов хорд, бисекции и метода Ньютона для расчета корня. - корень, высчитанный по методу ‘\*’, – количество итераций, затраченное на приближение корня методом ‘\*’, где ‘\*’ принимает значения: H – метод хорд, B – метод бисекции и N – метод Ньютона.

**Выводы.**

Проанализировав результаты применения метода Ньютона, можно сказать, что при расчете данной функции он дает очень хорошие результаты, и сходится за минимальное среди рассмотренных методов число итераций, которое соответствует теоретическому значению порядка.

По результатам эксперимента по определению обусловленности метода Ньютона можно оценить значение абсолютной обусловленности как значение приметно равное , что говорит об очень хорошей обусловленности метода.

Проанализировав данные вычислительного эксперимента по трем методам: Ньютона, хорд и бисекций, можно сделать вывод, что для данной функции метод Ньютона сходится наиболее быстрым образом и имеет преимущество перед рассмотренными в сравнении методами. Из недостатков метода можно выделить необходимость расчета аналитической формулы первой производной функции, и значения второй производной в точке.

Таблица 3 – Сравнение методов Ньютона, хорд и бисекции

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Значение eps | Значение delta | Значение | Значение | Значение | Значение | Значение | Значение |
| 0.1 | 0.001 | 0.875 | 0.858529 | 0.872102 | 2 | 3 | 2 |
| 0.01 | 0.001 | 0.863281 | 0.865001 | 0.867114 | 6 | 4 | 3 |
| 0.001 | 0.001 | 0.867676 | 0.867002 | 0.867114 | 9 | 5 | 3 |
| 0.0001 | 0.001 | 0.866943 | 0.867002 | 0.867114 | 9 | 5 | 3 |
| 0.00001 | 0.001 | 0.866943 | 0.867002 | 0.867114 | 9 | 5 | 3 |
| 0.000001 | 0.001 | 0.866943 | 0.867002 | 0.867114 | 9 | 5 | 3 |
| 0.1 | 0.0001 | 0.875 | 0.858435 | 0.872037 | 2 | 3 | 2 |
| 0.01 | 0.0001 | 0.863281 | 0.864908 | 0.867108 | 6 | 4 | 3 |
| 0.001 | 0.0001 | 0.867676 | 0.866953 | 0.867108 | 9 | 6 | 3 |
| 0.0001 | 0.0001 | 0.867126 | 0.867086 | 0.867108 | 12 | 8 | 3 |
| 0.00001 | 0.0001 | 0.867081 | 0.867086 | 0.867078 | 13 | 8 | 4 |
| 0.000001 | 0.0001 | 0.876081 | 0.867086 | 0.867078 | 13 | 8 | 4 |
| 0.1 | 0.00001 | 0.875 | 0.858425 | 0.872045 | 2 | 3 | 2 |
| 0.01 | 0.00001 | 0.863281 | 0.864907 | 0.867107 | 6 | 4 | 3 |
| 0.001 | 0.00001 | 0.867676 | 0.866949 | 0.867107 | 9 | 6 | 3 |
| 0.0001 | 0.00001 | 0.867126 | 0.867075 | 0.867107 | 12 | 8 | 3 |
| 0.00001 | 0.0001 | 0.867092 | 0.867084 | 0.86786 | 16 | 9 | 4 |
| 0.000001 | 0.0001 | 0.867084 | 0.867084 | 0.86786 | 17 | 9 | 4 |

Приложение А

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <iostream>

#include <conio.h>

double delta;

#ifndef \_\_NEWTON

#define \_\_NEWTON

#endif

#ifndef M\_PI

#define M\_PI 3.14159265358979323846

#endif // !M\_PI

#ifndef FF(x)

#define FF(x) ( (M\_PI \* pow(x, M\_PI + 7) + 2 \* M\_PI\*pow(x, M\_PI + 3) + M\_PI \* pow(x, M\_PI - 1) + 4 \* pow(x, 3)) / (pow(x, 8) + 2 \* pow(x, 4) + 1) )

#define FFF(x) ( (PI2\*pow(x, M\_PI + 10) - M\_PI \* pow(x, M\_PI + 10) + 3 \* PI2\*pow(x, M\_PI + 6) - 3 \* M\_PI\*pow(x, M\_PI + 6) + 3 \* PI2\*pow(x, M\_PI + 2) - 3 \* M\_PI\*pow(x, M\_PI + 2) - 20 \* pow(x, 6) + PI2 \* pow(x, M\_PI - 2) - M\_PI\*pow(x,M\_PI-2) + 12 \* pow(x, 2)) / (pow(x, 12) + 3 \* pow(x, 8) + 3 \* pow(x, 4) + 1) )

#endif

extern double F(double);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция F(X) , задаваемая пользователем \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifdef \_\_NEWTON

extern double F1(double);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Производная функции F(X) , задаваемая пользователем \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#endif

#ifdef \_\_ITER

extern double PHI(double);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция PHI(X) , задаваемая пользователем \*/

/\* Данная функция используется в методе \*/

/\* простых итераций \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#endif

double Round(double, double);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция Round (X, Delta) , предназначена для округления \*/

/\* X с точностью Delta \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double BISECT(double, double, double, int&);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция BISECT предназначена для решения уравнения F(X)=0 \*/

/\* методом деления отрезка пополам. Использованы обозначения: \*/

/\* Left - левый конец промежутка \*/

/\* Right - правый конец промежутка \*/

/\* Eps - погрешность вычисления корня уравнения; \*/

/\* N - число итераций \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double ITER(double, double, int&);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция ITER предназначена для решения уравнения F(X)=X \*/

/\* методом простой итерации. Использованы обозначения: \*/

/\* X0 - начальное приближение корня \*/

/\* Eps - погрешность вычисления корня уравнения; \*/

/\* N - число итераций \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double HORDA(double, double, double, int&);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция HORDA предназначена для решения уравнения F(x)=0 \*/

/\* методом хорд. Использованы обозначения: \*/

/\* Left - левый конец промежутка \*/

/\* Right - правый конец промежутка \*/

/\* Eps - погрешность вычисления корня уравнения; \*/

/\* N - число итераций \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double NEWTON(double, double, int&);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция NEWTON предназначена для решения уравнения F(X)=0 \*/

/\* методом касательных. Использованы обозначения: \*/

/\* X - начальное приближение корня \*/

/\* Eps - погрешность вычисления корня уравнения; \*/

/\* N - число итераций \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double Round(double X, double Delta) {

if (Delta <= 1E-9) {

puts("Неверное задание точности округления\n");

exit(1);

}

if (X > 0.0) {

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

} else {

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

}

}

double F(double x) {

// функция f(x)

extern double delta;

double s;

long S;

s = pow(x, M\_PI) - 1/(pow(x,4)+1);

s = Round(s, delta);

return s;

}

double F1(double x) {

// функция f'(x)

double f = (M\_PI \* pow(x, M\_PI + 7) + 2 \* M\_PI\*pow(x, M\_PI + 3) + M\_PI \* pow(x, M\_PI - 1) + 4 \* pow(x, 3)) / (pow(x, 8) + 2 \* pow(x, 4) + 1);

return f;

}

double PHI(double x) {

// функция φ(x) - для метода простых итераций

return x;

}

double BISECT(double Left, double Right, double Eps, int &N) {

double E = fabs(Eps) \* 2.0;

double FLeft = F(Left);

double FRight = F(Right);

double X = 0.5 \* (Left + Right);

double Y;

if (FLeft \* FRight > 0.0) {

puts("Неверное задание интервала\n");

exit(1);

}

if (Eps <= 0.0) {

puts("Неверное задание точности\n");

exit(1);

}

if (FLeft == 0.0) {

return Left;

}

if (FRight == 0.0) {

return Right;

}

for (N = 0; Right - Left >= E; N++) {

X = 0.5 \* (Right + Left); // вычисление середины отрезка

Y = F(X);

if (Y == 0.0) {

return X;

}

if (Y \* FLeft < 0.0) {

Right = X;

} else {

Left = X;

FLeft = Y;

}

}

return X;

}

#ifdef \_\_ITER

double ITER(double X0, double Eps, int &N) {

extern double PHI(double);

if (Eps <= 0.0) {

puts("Неверное задание точности\n");

exit(1);

}

double X1 = PHI(X0);

double X2 = PHI(X1);

for (N = 2;

(X1 - X2) \* (X1 - X2) > fabs((2 \* X1 - X0 - X2) \* Eps);

N++) {

X0 = X1;

X1 = X2;

X2 = PHI(X1);

}

return X2;

}

#endif

#ifdef \_\_NEWTON

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

extern double F1(double);

double Y, Y1, DX, Eps0;

N = 0;

double m1 = 1.154884, // наименьшее значение модуля 1-ой производной

M2 = 7.268115; // наибольшее значение модуля 2-ой производной

Eps0 = sqrt(2 \* m1 \* Eps / M2);

do {

Y = F(X);

if (Y == 0.0) {

return X;

}

Y1 = F1(X);

if (Y1 == 0.0) {

puts("Производная обратилась в ноль\n");

exit(1);

}

DX = Y / Y1;

X -= DX;

N++;

} while (fabs(DX) > Eps0);

return X;

}#endif

double HORDA(double Left, double Right, double Eps, int &N) {

double FLeft = F(Left);

double FRight = F(Right);

double X, Y;

if (FLeft \* FRight > 0.0) {

puts("Неверное задание интервала\n");

exit(1);

}

if (Eps <= 0.0) {

puts("Неверное задание точности\n");

exit(1);

}

N = 0;

if (FLeft == 0.0) {

return Left;

}

if (FRight == 0.0) {

return Right;

}

do {

X = Left - (Right - Left) \* FLeft / (FRight - FLeft);

Y = F(X);

if (Y == 0.0) {

return X;

}

if (Y \* FLeft < 0.0) {

Right = X;

FRight = Y;

} else {

Left = X;

FLeft = Y;

}

N++;

} while (fabs(Y) >= Eps);

return X;

}

#define FF(x) ( (M\_PI \* pow(x, M\_PI + 7) + 2 \* M\_PI\*pow(x, M\_PI + 3) + M\_PI \* pow(x, M\_PI - 1) + 4 \* pow(x, 3)) / (pow(x, 8) + 2 \* pow(x, 4) + 1) )

int main()

{

int k\_B,k\_H,k\_N;

long int s;

float a1, b1, eps1, delta1;

double a, b, eps, x\_B, x\_H, x\_N;

a = 0.5;

b = 1.25;

double x = 0.867086;

printf("eps\t\tdelta\t\ta\t\tb\t\tx\_B\t\tx\_H\t\tx\_N\t\t\tk\_B\tk\_H\tk\_N\tDx\t\tc\n");

for (delta = 0.1; delta >= 0.000001; delta /= 10)

{

for (eps = 0.1; eps >= 0.000001; eps /= 10)

{

x\_B = BISECT(a, b, eps, k\_B);

x\_H = HORDA(a, b, eps, k\_H);

x\_N = NEWTON(b, eps, k\_N);

printf("%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%d\t%d\t%d\t%lf\t%d\n", eps, delta, a, b, x\_B, x\_H, x\_N, k\_B, k\_H, k\_N, abs(x-x\_N), eps>=abs(x - x\_N));

}

}

return 0;

}