**Условие задачи**

Двумя методами для N значений погрешности e (0,1; 0,01;0,001;..1e-N, 1≤N≤10) вычислить значение корня для двух заданных функций (№ и №+3 по табл.2) на отрезке [A, B] (0,2) и вывести их в виде таблицы

|  |  |  |
| --- | --- | --- |
| **№** | **Функция** | **Методы** |
| **2** | ;  1,08306 | 1)Метод деления отрезка пополам  2) Метод Ньютона |
| **5** | ;  0,38462 | 1)Метод деления отрезка пополам  2) Метод Ньютона |

**Метод деления пополам**

На каждом шаге отрезок уменьшается вдвое. В начале каждой итерации находим середину нового отрезка [a , b]. Затем следует определить, с какой стороны от середины отрезка x находится корень x\*. Для этого достаточно сравнить знаки f (x) и f (b) или знаки f (x) и f (a). Если знаки f (x) и f (b) не совпадают, то это означает, что f (x) пересекает ось x на правом полуотрезке [x, b]. Заменяем a на x. Если же знаки f (x) и f (b) совпадают, то f (x) пересекает ось x на левом полуотрезке [a, x]. Заменяем b на x. Итак, в результате выполнения итерации отрезок [a, b] как и прежде, содержит единственный корень, но его длина стала меньше в два раза. Вычисления следует прекратить, если на очередном шаге длина отрезка [a, b] станет меньше ε

**Метод Ньютона**

Пусть имеется начальное приближение к корню, которое обозначим x1 = середине отрезка. Новое приближение к корню, которое мы будем называть следующим приближением, x2 получим по следующей расчетной формуле: . Теперь заменим значение начального приближения x1 на значение только что полученного приближения x2. Каждое такое улучшение приближения к корню за счет вычисления следующего приближения называется итерацией. Обычно считают, что требуемая точность достигнута, если после вычисления x2 при выполнении очередной итерации соблюдается условие |x2–x1|< e При выполнении этого неравенства итерационный процесс уточнения корня следует прекратить и в качестве искомого приближенного значения корня взять x2.Смысл условий сходимости метода Ньютона состоит в том, что 1)начальное приближение x1, используемое при выполнении первой итерации, должно быть не слишком далеко от корня, а 2) производная f’(x), которая находится в знаменателе, должна изменяется на отрезке [a, b] не очень быстро и не обращаться в ноль ни в одной точке отрезка [a, b]. Мы будем считать, что они выполняются. Метод Ньютона является наиболее быстрым среди численных методов вычисления корня функционального уравнения. На практике необходимая точность достигается буквально после выполнения нескольких (не более 10) итераций. Для вычисления входящей в формулу (2) производной f‘(x) можно воспользоваться выражением для ее приблизительного вычисления при малом значении

**Комментарии**

Примечание: дополнительная функция создана в одном файле с расширением .h

Комментарии расставлены в программе.

**Результат работы программы**

Отрезок [0.1,1.9]

Функция 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод  Точность | Деление пополам | | Метод Ньютона | |
| Корень | Итерации | Корень | Итерации |
| 0.1 | 1.1 | 5 | 1.1 | 2 |
| 0.5 | 1.4 | 2 | 1.1 | 2 |
| 0.01 | 1.08 | 8 | 1.08 | 2 |
| 0.05 | 1.08 | 6 | 1.08 | 2 |
| 0.001 | 1.083 | 11 | 1.083 | 3 |
| 0.005 | 1.081 | 9 | 1.083 | 2 |
| 0.0001 | 1.0831 | 15 | 1.0831 | 3 |
| 0.0005 | 1.0831 | 12 | 1.0831 | 3 |
| 0.00001 | 1.08308 | 18 | 1.08307 | 3 |
| 0.00005 | 1.08308 | 16 | 1.08307 | 3 |

Функция 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод  Точность | Деление пополам | | Метод Ньютона | |
| Корень | Итерации | Корень | Итерации |
| 0.1 | 0.4 | 5 | 0.4 | 2 |
| 0.5 | 0.6 | 2 | 0.4 | 2 |
| 0.01 | 0.39 | 8 | 0.38 | 3 |
| 0.05 | 0.41 | 6 | 0.38 | 2 |
| 0.001 | 0.384 | 11 | 0.385 | 3 |
| 0.005 | 0.385 | 9 | 0.385 | 3 |
| 0.0001 | 0.3846 | 15 | 0.3846 | 3 |
| 0.0005 | 0.3843 | 12 | 0.3846 | 3 |
| 0.00001 | 0.38462 | 18 | 0.38462 | 4 |
| 0.00005 | 0.38463 | 16 | 0.38462 | 3 |

Отрезок [0.3,1.1]

Функция 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод  Точность | Деление пополам | | Метод Ньютона | |
| Корень | Итерации | Корень | Итерации |
| 0.1 | 1.1 | 4 | 1.1 | 2 |
| 0.5 | 0.7 | 1 | 1.1 | 2 |
| 0.01 | 1.08 | 7 | 1.08 | 3 |
| 0.05 | 1.08 | 5 | 1.08 | 2 |
| 0.001 | 1.084 | 10 | 1.083 | 3 |
| 0.005 | 1.084 | 8 | 1.083 | 3 |
| 0.0001 | 1.0831 | 13 | 1.0831 | 4 |
| 0.0005 | 1.0832 | 11 | 1.0831 | 3 |
| 0.00001 | 1.08307 | 17 | 1.08307 | 4 |
| 0.00005 | 1.08306 | 14 | 1.08307 | 4 |

Функция 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод  Точность | Деление пополам | | Метод Ньютона | |
| Корень | Итерации | Корень | Итерации |
| 0.1 | 0.3 | 4 | 0.4 | 2 |
| 0.5 | 0.7 | 1 | 0.4 | 2 |
| 0.01 | 0.38 | 7 | 0.38 | 2 |
| 0.05 | 0.38 | 5 | 0.38 | 2 |
| 0.001 | 0.385 | 10 | 0.385 | 3 |
| 0.005 | 0.384 | 8 | 0.385 | 2 |
| 0.0001 | 0.3847 | 13 | 0.3846 | 3 |
| 0.0005 | 0.3848 | 11 | 0.3846 | 3 |
| 0.00001 | 0.38463 | 17 | 0.38462 | 3 |
| 0.00005 | 0.38462 | 14 | 0.38462 | 3 |

**Вывод:** на основе данных, полученных в таблице, можно сделать вывод, что хоть метод Ньютона по числу итерации самый быстрый, но при высокой точности (0.00005) является иногда менее точным, чем метод бисекции, однако в иных случаях больше подходит для вычислений, поскольку в общем быстрее и достаточно точный

**Код**

**Программа**

#include <cstdio>

#include <cmath>

#include <cstdlib>

#include "methodsFunc.h"

**int main() {**

printf("Enter A, B\n"); // Запрос ввода концов отрезка

int n;

double a,b;

try {

scanf("%lf", &a); // Чтение левого края отрезка с клавиатуры

if (a<=0) throw 1; if (a>=2) throw 2; // Проверка на аномалии (левый конец отрезка не должен быть меньше 0 или равен/больше 2) с выдачей кода ошибки

scanf("%lf", &b);

if (b<=0) throw 3; if (b>=2) throw 4;// Проверка на аномалии (правый конец отрезка не должен быть меньше 0 или равен/больше 2) с выдачей кода ошибки

if (b<a) throw 5; // Проверка на аномалии (правый конец отрезка не должен быть меньше левого) с выдачей кода ошибки

fflush(stdin);

printf("Enter number of tries\n"); // Запрос на ввод количества погрешностей

scanf("%d", &n); if (n<1) throw 6; if (n>10) throw 7; // Проверка на аномалии (количество погрешностей не должно быть меньше 1 или больше 10) с выдачей кода ошибки

double e[n]; // Массив из погрешностей

for (int i = 0; i < n; i++) { // Заполнение массива погрешностей

printf("Accuracy[%d]=", i + 1);

scanf("%lf", &e[i]);

if (floor(e[i]\*pow(10,10))==0) throw 8 + i; // Проверка на аномалии (Погрешность не должна быть слишком маленькой) с выдачей кода ошибки

fflush(stdin);

}

printf("For the First function:\n");

**TableStart(2, a, b); // Начало таблицы**

for (int i = 0; i < n; i++) { // Поиск корня

int n1 = 0;

int n2 = 0;

**double x1 = FindRootDiv(a, b, e[i], func2, &n1); // Деление пополам**

**double x2 = FindRootNewt(a, b, e[i], func2, &n2); // Метод Ньютона**

**TableCell(e[i], x1, n1, x2, n2); // Вывод в таблицу**

}

printf("For the Second function:\n");

**TableStart(5, a, b); // Заголовок таблицы**

for (int i = 0; i < n; i++) { // Поиск корня

int n1 = 0;

int n2 = 0;

**double x1 = FindRootDiv(a, b, e[i], func5, &n1); // Деление пополам**

**double x2 = FindRootNewt(a, b, e[i], func5, &n2); // Метод Ньютона**

**TableCell(e[i], x1, n1, x2, n2); // Вывод в таблицу**

}

}

catch (int errNum) { // Расшифровка кодов ошибок

switch (errNum){

default: // По умолчанию (код больше 7)

errNum-=7;

printf("Accuracy[%d] is too little\n",errNum);

break;

case 1: // Код ошибки 1

printf("\'A\' should be bigger than 0\n");

break;

case 2: // Код ошибки 2

printf("\'A\' should be lesser than 2\n");

break;

case 3: // Код ошибки 3

printf("\'B\' should be bigger than 0\n");

break;

case 4: // Код ошибки 4

printf("\'B\' should be lesser than 2\n");

break;

case 5: // Код ошибки 5

printf("\'A\' should not be bigger than \'B\'\n");

break;

case 6: // Код ошибки 6

printf("\'N\' should be bigger than 1\n");

break;

case 7: // Код ошибки 7

printf("\'N\' should be lesser than 10\n");

break;

}

}

printf("------------------------------------------------------------\nPress ENTER"); getc(stdin);

return 0;

**}**

**Модуль**

**typedef double (\*func)(double); // Указатель на функцию типа double с принимаемым значением double**

**double func2(double x) // Функция 2 варианта**

{

return (

((double)1/(double)10)\*exp(-pow(cos(x),(double)2))+sqrt(x/(double)2)/log(x+(double)1)-x

);

}

**double func5(double x) // Функция 5 варианта**

{

return (

(double)1/(sqrt((double)5)+sin((double)0.1\*x)+log((double)1+x))-x

);

}

**double FindRootDiv(double a, double b, double e, func f,int \*n) // Метод деления отрезка**

{

double xn,FB,FX;

if (f(a)\*f(b)>0) // Проверка на наличие корня на отрезке (Разные знаки на концах отрецка)

return -1; // Если нет - возращаем -1

else do // Цикл с постусловием

{

xn = (b + a) / (double) 2; // Середина отрезка

FB = f(b); // F(b) правый край

FX = f(xn); // F(X) левый сходящийся край

if (FX \* FB < 0) a = xn; // Сравнение знаков F(x) и F(b)

else b = xn;

(\*n)++;

} while (fabs(a - b) >= e && FX != 0); // Ищем, пока не пересечем Ox или пока отрезок не будет меньше точности(погрешности)

return xn;

}

**double FindRootNewt(double a, double b, double e, func f, int \*n) // Метод Ньютона**

{

double x0,xn, FX, dFX;

if (f(a)\*f(b)>0) // Проверка на наличие корня на отрезке (Разные знаки на концах отрецка)

return -1; // Если нет - возращаем -1

else {

x0 = (a + b) / 2; // Начальное приближение корня равно половине отрезка

FX = f(x0); // F(x0) - значение функции в точке x0

dFX = (f(x0 + e/2) - f(x0 - e/2))/e; // dF(x0) - производная функции F(x0)

xn = x0 - FX / dFX; // Считаем первое приближение

(\*n)++;

do //Цикл с постусловием

{

x0 = xn; // Начальное приближение корня = первому приближению

FX = f(x0); // F(x0) - значение функции в точке x0

dFX = (f(x0 + e/2) - f(x0 - e/2))/e; // dF(x0) - производная функции F(x0)

xn = x0 - FX / dFX; // Непосредственно формула Ньютона

(\*n)++;

} while (fabs(x0 - xn) > e); // Ищем, пока отрезок не будет меньше точности(погрешности)

return xn;

}

}

**void TableStart(int n,double a,double b) // Заголовок таблицы с результатами для удобного вывода**

{

printf("Line segment [%.5f, %.5f]\n",a,b);

printf("|Function N%d | Segment division |Newton's method |\n",n); printf("|Accuracy |Root |Iterations |Root |Iterations |\n");

}

**void TableCell(double e,double x1,int n1, double x2, int n2) // Таблица с результатами для удобного вывода**

{

int z = ceil(fabs(log(e)/log(10)));

char\* r1 = new char[z+2];

char\* r2 = new char[z+2];

if (x1==-1) r1 = (char\*)"No root ";

else sprintf(r1,"%.\*f",z,x1);

if (x2==-1) r2 = (char\*)"No root ";

else sprintf(r2,"%.\*f",z,x2);

printf("|%12.\*f|%12s|%9d|%12s|%9d|\n",z,e,r1,n1,r2,n2);

delete []r1; delete []r2;

}