**Задание 4**

**«Процедуры и функции в качестве параметров».**

**Оглавление**

[Постановка задачи 2](#__RefHeading___Toc979_724668214)

[Решение 2](#__RefHeading___Toc981_724668214)

[Методы 3](#__RefHeading___Toc983_724668214)

[Дихотомия или метод половинного деления. 3](#__RefHeading___Toc985_724668214)

[Метод итераций. 3](#__RefHeading___Toc987_724668214)

[Метод Ньютона. 3](#__RefHeading___Toc989_724668214)

[Вычисление машинного эпсилон 4](#__RefHeading___Toc991_724668214)

[Описание функций программы 4](#__RefHeading___Toc993_724668214)

[Протокол 5](#__RefHeading___Toc995_724668214)

[Вывод 7](#__RefHeading___Toc997_724668214)

# **Постановка задачи**

Составить программу на языке Си с процедурами решения трансцендентных алгебраических уравнений различными численными методами (итераций, Ньютона и половинного деления — дихотомии). Нелинейные уравнения оформить как параметры-функции, разрешив относительно неизвестной величины в случае необходимости.

Вариант 23

Уравнение *3x - 4ln(x) - 5= 0*

Отрезок [2, 4]

Вариант 24

Уравнение  *cos(2/x)-2sin(1/x)+1/x=0*

Отрезок [1, 2]

# Решение

Всё решение сводится к тому, чтобы запрограммировать на языке Си три функции, ищущие корни заданных трансцендентных уравнений на заданных отрезках. Все функции, реализующие вычисление корней различными методами представляют собой различные итерационные процессы, в ходе которого последовательно вычисляется приближенные значения корня уравнения на заданном отрезке.

Основная проблема заключается в точности вычислений. Поэтому в решении данной задачи используется значение машинного эпсилон.

Машинное эпсилон — это максимальная относительная погрешность для конкретной процедуры округления, это такое числовое значение, меньше которого невозможно задавать относительную точность для любого алгоритма, возвращающего вещественные числа. В данной программе оно будет использоваться как признак того, что вычисленное значение уже достаточно точное, и дальнейшие вычисления уже не целесообразны т. к. дальнейшие числа являются одинаковыми с точки зрения машинной арифметики.

# Методы

## Дихотомия или метод половинного деления.

Уравнение имеет действительные корни на заданном отрезке, если функция на нем непрерывна и её значения на концах отрезка имеют разные знаки. Последовательно будем делить отрезок пополам и выбирать ту половину, на которой лежит корень (то есть значения функции на концах которого имеют разные знаки), пока не будет достигнута заданная точность.

## Метод итераций.

Уравнение f(x)=0 преобразуем в виде x = F(x). Выберем на заданном отрезке его середину x0 в качестве начального приближения и построим последовательность: x1=F(x0 ), x2=F(x1 ), ..., xn= F(xn-1 ). Процесс итераций сходится если |f’(x)| < 1 на отрезке, и увеличивая n, можно получить приближение, сколь угодно мало отличающееся от истинного значения корня.

## Метод Ньютона.

Метод Ньютона можно применять, когда известно, что функция на заданном отрезке имеет один корень, причем первая и вторая производные на этом отрезке определены, непрерывны и сохраняют постоянные знаки. Возьмем x0 - середину отрезка [a,b] и проведем в точке P0 {x0 ,f(x0 )} графика функции касательную к кривой y=f(x) до пересечения с осью Ox. Абсциссу x1 точки пересечения можно взять в качестве приближенного значения корня. Проведя касательную через новую точку P1 {x1 ,f(x1 )} и находя точку ее пересечения с осью Ox, получим второе приближение корня x2 . Получившаяся последовательность сходится, если |f(x) · f ''(x)| < (f ’(x))2.

# Вычисление машинного эпсилон

Делим e равную единице пополам пока не получится так, что мы не можем отличить 1 + e\2 и 1. Если так случилось, значит, разница на предыдущем шаге и есть машинное эпсилон.

double eps()

{

double e = 1.0f;

while (1.0f + e / 2 > 1.0f)

e /= 2;

return e;

}

# Описание функций программы

|  |  |  |
| --- | --- | --- |
| Название функции | Входные аргументы | Описание функции |
| eps | - | Функция считает машинный epsilon, методом, описанным выше, а именно сравнивая 1+ε/2 и 1. Пока выражение 1 < 1 + ε/2 возвращает true, функция делит epsilon пополам. |
| F24 | double x | Возвращает функцию 24 варианта arccos(x)-cos(2/x)-2sin(1/x)+1/x |
| pF24 | double x | Возвращает производную функции 24 варианта cos(2/x)-2sin(1/x)+1/x |
| F23 | double x | Возвращает функцию 23 варианта 3\*x-4\*lnx-5 |
| pF23 | double x | Возвращает производную функции 23 варианта 3\*x-4\*lnx-5 |
| iterF23 | double x | Возвращает функцию эквивалентную функции 3\*x-4\*lnx-5 |
| iterF24 | double x | Возвращает функцию эквивалентную функции cos(2/x)-2sin(1/x)+1/x |
| Dihotomy | double (\*f)(double), double a, double b | Поиск значения функции методом дихотомии |
| Iterations | double (\*f)(double), double x | Поиск значения функции методом итераций |
| Newton | double (\*f)(double), double (\*fp)(double), double x | Поиск значения функции методом Ньютона |

# Протокол

polina@pelis:~$ cat > kurs4.c

#include <stdio.h>

#include <math.h>

double eps()

{

double e = 1.0f;

while (1.0f + e / 2 > 1.0f)

e /= 2;

return e;

}

double F24(double x)

{

return cos(2/x) - 2\*sin(1/x) + 1/x;

}

double pF24(double x)

{

return (2\*sin(2/x))/(x\*x)+(2\*cos(1/x))/(x\*x)-1/(x\*x);

}

double F23(double x)

{

return 3 \* x - 4 \* log(x) - 5;

}

double pF23(double x)

{

return 3 - 4 / x;

}

double iterF23(double x)

{

return (5 + 4 \* log(x)) / 3;

}

double iterF24(double x)

{

return (double) 2 / (acos(2 \* sin(1/x) - (double) 1 / x)) ;

}

double Dihotomy(double (\*f)(double), double a, double b)

{

double l = a;

double r = b;

while (fabs(r - l) > eps())

{

if (f(l) \* f((l + r) / 2) >= 0)

{

l = (l + r) / 2;

}

else

{

r = (l + r) / 2;

}

}

return (l + r) / 2;

}

double Iterations(double (\*f)(double), double x)

{

double prev = x;

double cur = f(x);

while (fabs(cur - prev) > eps())

{

prev = cur;

cur = f(prev);

}

return cur;

}

double Newton(double (\*f)(double), double (\*fp)(double), double x)

{

double prev = x;

double cur = x - (f(x) / fp(x));

while (fabs(cur - prev) > eps())

{

prev = cur;

cur = prev - (f(prev) / fp(prev));

}

return cur;

}

int main()

{

double a = 2.0, b = 4.0;

printf("----------------------------------------------------------------------------------------------------\n");

printf("| № | Уравнение | Метод дихотомии | Метод итераций | Метод Ньютона |\n");

printf("----------------------------------------------------------------------------------------------------\n");

printf("| 23 | 3x-4lnx-5=0 | %.16f | %.16f | %.16f |\n", Dihotomy(F23, a, b), Iterations(iterF23, (b + a) / 2), Newton(F23, pF23, (b + a) / 2));

a = 1.0, b = 2.0;

printf("----------------------------------------------------------------------------------------------------\n");

printf("| 24 | cos(2/x)-2sin(1/x)+1/x=0 | %.16f | %.16f | %.16f |\n", Dihotomy(F24, a, b), Iterations(iterF24, (b + a) / 2), Newton(F24, pF24, (b + a) / 2));

printf("----------------------------------------------------------------------------------------------------\n");

}

^C

polina@pelis:~$ gcc kurs4.c -lm

polina@pelis:~$ ./a.out

----------------------------------------------------------------------------------------------------

| № | Уравнение | Метод дихотомии | Метод итераций | Метод Ньютона |

----------------------------------------------------------------------------------------------------

| 23 | 3x-4lnx-5=0 | 3.2299594397279288 | 3.2299594397279279 | 3.2299594397279283 |

----------------------------------------------------------------------------------------------------

| 24 | cos(2/x)-2sin(1/x)+1/x=0 | 1.8756173924904571 | 1.8756173924904571 | 1.8756173924904571 |

----------------------------------------------------------------------------------------------------

polina@pelis:~$

# Вывод

После генерации таблицы значений заданной функции можно увидеть, что значения корней уравнений, вычисленных разными способами совпадают или практически совпадают до 15-16 знака после запятой. Из-за того, что методы вычисления корней имеют различную «скорость» приближения к точному значению корня на данном отрезке, и один алгоритм достигает точки останова быстрее другого, а так же вещественные числа имеют диапазон представления в памяти компьютера, что неизбежно приводит к тому, что в вычислениях в окрестности границ этого диапазона возникают погрешности. Отсюда и небольшие различия в значениях корней.

Существуют более совершенные способы решения задач, подобных данной, но вычисления корней с помощью данных методов могут быть использованы так же и в приложении простого калькулятора для решения подобных уравнений.