1. Цель работы

Научиться решать нелинейные уравнения, изучить методы Ньютона, хорд и дихотомии, написать решения на c++ и scilab и сравнить полученные результаты.

2. Теория

2.1 Метод хорд

Метод хорд основан на итерационной формуле .

$$x_{i+1} = x_{i-1} - \frac{f(x_{i-1}) \cdot (x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}.$$

Повторять вычисления необходимо, до тех пор, пока разница между текущим и предыдущим вычислением больше чем необходимая точность.

2.2 Метод Ньютона

Метод Ньютона или метод касательных основан на итерационной формуле метода Ньютона.

$$X_1 = X_0 - \frac{F(X_0)}{F'(X_0)}$$

На каждой итерации корень уточняется и процесс продолжается до тех пор, пока корень не будет находится в пределах заданной точности. Данная формула может давать ошибки при неправильно заданном начальном значении x0.

2.3 Метод дихотомии

Допустим на всём отрезке [a, b] функция f(x) определена, имеет ровно один корень и на границах отрезка имеет разные знаки. То в таком случае для уточнения корня мы можем применить метод дихотомии, который заключается в том, что на каждой итерации находится центр текущего отрезка, обозначим за c, если f(a) * f(c) > 0, то на следующей итерации рассматривается отрезок [c, b], иначе на следующей итерации рассматривается отрезок [a, c]. И так повторяется до тех пор, пока длинна отрезка не станет меньше заданной точности.

3. Ход работы

3.1 Реализация на c++

Первым этапом в работе было изучение теоретического материала. После, используя блок-схемы, примеры исходного кода и опираясь на математические

формулы, были реализованы заданные методы на языке программирования C++. В методе Ньютона были убраны переменные, которые были необходимы для кеширования данных. Так как вычисления функции моего варианта не являются требовательными для компьютера, я заменил их на повторные вычисления, с целью улучшить исходный код и его внешний вид.

```
#include <iostream>
#include <cmath>
#include <cassert>
#define         EPS             0.0005
#define assertm(exp, msg) assert(((void)msg, exp))


double func(double x);
double func1(double x);
double func2(double x);
double dihotomii_method(double start, double end);
double hord_method(double start, double end);
double newton_method(double x0);


int main() {
        // x^3 + x + 3 = 0;
        double start = -2, end = -1;

        double newton_method_answer = newton_method(start);
        std::cout << "newton_method_answer: "<< newton_method_answer << std::endl;


        double dihotomii_method_answer = dihotomii_method(start, end);
        std::cout << "dihotomii_method_answer: "<< dihotomii_method_answer <<
std::endl;

        double hord_method_answer = hord_method(start, end);
        std::cout << "hord_method_answer: "<< hord_method_answer << std::endl;

        return 0;
}

double func(double x) { return x*x*x + x + 3; }

double func1(double x) { return 3*x*x + 1; } // производная первого порядка

double func2(double x) { return 6*x; }  // производная второго порядка

double newton_method(double x0) {
        assertm(fabs(func(x0)*func2(x0)/(func1(x0)*func1(x0))) < 1, "Процесс
расходится x0 выбран не правильно.");
```

```
        double xi = x0;
        while (fabs(func(xi)) > EPS) {
                xi = xi - func(xi)/func1(xi);
        }
        return xi;
}

double hord_method(double start, double end) {
        // [start, end] - где мы ищем
        assertm(func(start) * func(end) < 0, "Интервал [a, b] выбран не
правильно");
        double a = start, b = end, c;
        while (fabs(b - a) > EPS) {
                c = a - (b-a) * func(a) / (func(b) - func(a));
                if (func(c) * func(a) > 0) {
                        b = c;
                } else {
                        a = c;
                }
        }
        return c;
}

double dihotomii_method(double start, double end) {
        // По сути это метод бинарного поиска,
        assertm(func(start) * func(end) < 0, "Интервал [a, b] выбран не
правильно");
        double middle;
        while (fabs(start-end) > EPS) {
                middle = (start + end) / 2;
                if (func(start)*func(middle) < 0) {
                        end = middle;
                } else {
                        start = middle;
                }
        }
        middle = (start+end) / 2;
        return middle;
}
```

Результат работы программы на c++:

```
newton_method_answer: -1.21341
dihotomii_method_answer: -1.21362
hord_method_answer: -1.21341
```

Выглядит так, как будто метод дихотомии даёт ошибку. На самом деле, оно находит решение с заданной точностью EPS=0.0005.

3.2 Реализация на scilab

Реализация на scilab является точной копией кода на c++.

```
function result = func(x)
        result = x*x*x + x + 3;
endfunction

function result = func1(x)
        result = 3*x*x + 1;
endfunction

function result = func2(x)
        result = 6*x;
endfunction

function result = newton_method(x0)
        xi = x0;
        while abs(func(xi)) > EPS
        xi = xi - func(xi)/func1(xi);
    end
    result = xi;
endfunction

function result = hord_method(start, end)
        a = start;
        b = end;
        while abs(b-a) > EPS
      c = a - (b-a)*func(a)/(func(b)-func(a));
          if (func(c)*func(a) > 0)
        b = c
      else
        a = c
      end
    end
    result = c;
endfunction

function result = dihotomii_method(start, end)
        while abs(start-end) > EPS
      middle = (start + end) / 2;
          if (func(start)*func(middle) < 0)
        end = middle;
      else
        start = middle;
      end
    end
        result = middle;
endfunction
```

```
EPS = 0.0005
start = -2
end = -1

newton_method_result = newton_method(start);
disp("newton_method_result")
disp(newton_method_result)

disp("hord_method_result")
disp(hord_method(start, end))

disp("dihotomii_method_result")
disp(dihotomii_method(start, end))
```

Результат работы:

```
EPS  =
   0.0005
start  =
 - 2.
end  =
 - 1.
newton_method_result
 - 1.2134121
hord_method_result
 - 1.2134117
dihotomii_method_result
 - 1.2133789
```

Здесь также решения находятся в пределах заданной точности EPS. Проблем с реализацией не было.

4. Выводы

Научился решать нелинейные уравнения, изучил методы Ньютона, хорд и дихотомии, напил решения на c++ и scilab. Все методы приводят к уточнению корня нелинейного уравнения на заданном промежутке до необходимой точности.