Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа № 2

по курсу «Численные методы».

Тема: «Метод решения нелинейных уравнений и систем нелинейных уравнений».

Студент: Вельтман Л.Я.
Группа: 80-307Б
Вариант: 7
Оценка:

Москва, 2020

Постановка задачи.

Реализовать методы для решения нелинейных уравнений:

1. Метод простых итераций
2. Метод Ньютона (касательных)

Описание методов.

Метод простой итерации. При использовании метода простой итерации уравнение (2.1) заменяется эквивалентным уравнением с выделенным линейным членом

$$x = \varphi(x) \tag{2.5}$$

Решение ищется путем построения последовательности

$$x^{(k+1)} = \varphi(x^{(k)}) \qquad k = 0,1,2,... \tag{2.6}$$

начиная с некоторого заданного значения $x^{(0)}$. Если $\varphi(x)$ - непрерывная функция, а $x^{(k)} \quad (k = 0,1,2,...)$ - сходящаяся последовательность, то значение $x^{(*)} = \lim\limits_{k\to\infty} x^{(k)}$ является решением уравнения (2.5).

Условия сходимости метода и оценка его погрешности определяются теоремой [2]:

Теорема 2.3. Пусть функция $\varphi(x)$ определена и дифференцируема на отрезке [a,b]. Тогда если выполняются условия:

1) $\varphi(x) \in [a,b] \quad \forall\, x \in [a,b]$,

2) $\exists q : |\varphi'(x)| \le q < 1 \quad \forall\, x \in (a,b)$,

то уравнение (2.5) имеет и притом единственный на [a,b] корень $x^{(*)}$;

к этому корню сходится определяемая методом простой итерации последовательность $x^{(k)} \quad (k = 0,1,2,...)$, начинающаяся с любого $x^{(0)} \in [a,b]$.

При этом справедливы оценки погрешности ($\forall\, k \in N$):

$$\left|x^{(*)} - x^{(k+1)}\right| \le \frac{q}{1-q}\left|x^{(k+1)} - x^{(k)}\right| \tag{2.7}$$

$$\left|x^{(*)} - x^{(k+1)}\right| \le \frac{q^{k+1}}{1-q}\left|x^{(1)} - x^{(0)}\right|.$$

Метод Ньютона. Для корректного использования данного метода необходимо, в соответствии с теоремой 2.2, определить поведение первой и второй производной функции $f(x)$ на интервале уточнения корня и правильно выбрать начальное приближение $x^{(0)}$.

Для функции $f(x) = e^{2x} + 3x - 4 = 0$ имеем:

$f'(x) = 2e^{2x} + 3$, $f''(x) = 4e^{2x}$ - положительные во всей области определения функции. В качестве начального приближения можно выбрать правую границу интервала $x^{(0)} = 0.6$, для которой выполняется неравенство (2.3):

$$f(0.6) f''(0.6) > 0$$

Дальнейшие вычисления проводятся по формуле (2.2), где

$f(x^{(k)}) = e^{2x^{(k)}} + 3x^{(k)} - 4$, $f'(x^{(k)}) = 2e^{2x^{(k)}} + 3$.

Итерации завершаются при выполнении условия $\left|x^{(k+1)} - x^{(k)}\right| < \varepsilon$.

Общая информация.

Данная работа позволяет решать нелинейные уравнения методами Ньютона и простых итераций. Что касается технических деталей реализации, программа написана на языке C++.

Запуск программы.

Чтобы воспользоваться программой, необходимо скомпилировать файл main.cpp и запустить полученный исполняемый файл:

g++ -std=c++11 main.cpp -o run

run.exe

Результаты.

Вариант 1.

0.0
1.0
0.001
2^x + x * x - 2
// a
// b
// eps
// funcs
	ANSWER:
	Newton's method
	x = 0.653483
	Iterations: 2
	Simple iterations method
	x = 0.653043
	Iterations: 33

Выводы.

Изучила методы простой итерации и Ньютона для решения нелинейных уравнений.

Исходный код.

```
/*
 2.1. Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

 */

#include <iostream>
#include <cmath>
#include <vector>

//const double e = 2.71828182846;
double foo(double x) {
    return (pow(2, x) + x * x - 2);
}


double firstDiff(double x) {
   return (pow(2, x) * log(2) + 2 * x);
}


double secondDiff(double x) {
   return (pow(2, x) * log(2) * log(2) + 2);
}


int NewtonsMethod(double a, double b, double eps, double& ans) {
```

```
    int k = 0;
    double xk_1 = /*0.6*/a + eps;
    double xk = xk_1;
    if (foo(a) * foo(b) < 0) {
        double criteria = eps;
        while (criteria >= eps) {
            if (foo(xk_1) * secondDiff(xk_1) > 0) {
                xk = xk_1;
                xk_1 = xk - (foo(xk)) / firstDiff(xk);
                ++k;
                criteria = abs(xk_1 - xk);
            } else {
                if (xk_1 < b) {
                    xk_1 += eps;
                } else {
                    throw "Try to find another length [a;b]";
                }
            }
        }
    } else {
        throw "Try to find another length [a;b]";
    }
    ans = xk_1;
    return k + 1;
}

double phi(double x) {
  return sqrt(2 - pow(2, x));
}


double phiDiff(double x) {
    return (-pow(2, x)* log(2)) / (2 * sqrt(2 - pow(2, x)));
}


int LyambdaSolution(double xk, double& xk_1, double eps) {
    int k = 0;
    double lyambda = eps;
    while (true)
        if (abs(1 - lyambda * firstDiff(xk_1)) < 1) {
            double criteria = eps;
            while (criteria >= eps) {
                xk = xk_1;
                xk_1 = phi(xk);
                ++k;
                criteria = abs(xk_1 - xk);
            }
            break;
        } else {
            lyambda += eps;
        }
```

```
    return k + 1;
}

int SimpleIterationsMethod(double a, double b, double eps, double& ans) {
    double x0 = (a + b) / 2, xk;
    double xk_1 = xk = x0;
    int k = 0;
    if (phi(x0) >= a && phi(x0) <= b) {
        if (abs(phiDiff(x0)) < 1) {
            double criteria = eps;
            while (criteria >= eps) {
                xk = xk_1;
                xk_1 = phi(xk);
                ++k;
                criteria = abs(xk_1 - xk);
            }
        } else {
            k = LyambdaSolution(xk, xk_1, eps);
        }
    } else {
        k = LyambdaSolution(xk, xk_1, eps);
    }
    ans = xk_1;
    return k + 1;
}
int main(int argc, const char * argv[]) {
    double accuracy = 0.001;
    double a = 0.0, b = 1.0, ans1 = 0.0, ans2 = 0.0;
    int iterNewton = NewtonsMethod(a, b, accuracy, ans1);
    int iterSimple = SimpleIterationsMethod(a, b, accuracy, ans2);

    std::cout << "ANSWER:" << std::endl;
    std::cout << "Newton's method" << std::endl;
    std::cout << "x = " << ans1 << std::endl;
    std::cout << "Iterations: " << iterNewton << std::endl;
    std::cout << "Simple iterations method" << std::endl;
    std::cout << "x = " << ans2 << std::endl;
    std::cout << "Iterations: " << iterSimple << std::endl;
    return 0;
}
```