Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа №1**

по дисциплине: «Методы оптимизации».

Вариант - 8

Выполнил:

студент 1 курса, гр. ИВТАСмд-11

Кондратьев Павел Сергеевич.

Проверил:

преподаватель кафедры ВТ

Валюх Вероника Валерьевна

г. Ульяновск, 2020

Оглавление

[Задание: 3](#_Toc54809360)

[Ход выполнения работы: 4](#_Toc54809361)

[Описание метода касательных нахождения экстремумов функции 4](#_Toc54809362)

[Алгоритм 5](#_Toc54809363)

[Пример 5](#_Toc54809364)

[Недостатки 6](#_Toc54809365)

[Реализация и сравнение эффективности программной реализации алгоритма 7](#_Toc54809366)

[Выводы: 11](#_Toc54809367)

[Онлайн реализации метода касательных 13](#_Toc54809368)

[Исследование реализации выбранного алгоритма в различных пакетах прикладных программ (MathCad; MATLAB). 15](#_Toc54809369)

[Список источников: 17](#_Toc54809370)

# Задание:

1. Составление библиотеки ссылок на исходники программ и онлайн реализации метода. Исследование реализации выбранного алгоритма в различных пакетах прикладных программ (Maple; MathCad; Mathematica; MATLAB)

2. Осмысление и описание метода оптимизации.

3. Реализация и сравнение эффективности реализации алгоритма в разных языках программирования (С++, С#, JAVA, язык по выбору). Исследование зависимости скорости выполнения от а) размерности задачи, б) языка.

4. Оформить отчёт о проделанной работе.

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Лабораторная 1**  **Методы поиска для функции одной переменной (одномерная оптимизация)**  **Методы последовательного поиска** | **Лабораторная 2**  **Методы оптимизации дифференцируемых функций**  **(Градиентные методы, прямые методы, методы первого порядка) \*** | **Лабораторная 3**  **Оптимизация на графах**  **(Поиски кратчайших путей, построение деревьев)** |
| 8 | метода Ньютона | Модифицированный метод наискорейшего спуска | Алгоритм Флойда |

**Лабораторная №1**

# Ход выполнения работы:

## Описание метода касательных нахождения экстремумов функции

Нахождение максимумов и минимумов на заданном интервале лежит в основе решения многих задач по оптимизации.

Метод Ньютона, алгоритм Ньютона (также известный как метод касательных) – это итерационный численный метод нахождения корня (нуля) заданной функции. Метод был впервые предложен английским физиком, математиком и астрономом Исааком Ньютоном (1643—1727).

Поиск решения осуществляется путём построения последовательных приближений и основан на принципах простой итерации. Метод обладает квадратичной сходимостью.

Модификацией метода является метод хорд и касательных. Также метод Ньютона может быть использован для решения задач оптимизации, в которых требуется определить ноль первой производной либо градиента в случае многомерного пространства.

Метод касательных основан на геометрическом смысле производной.

**Геометрическая интерпретация**

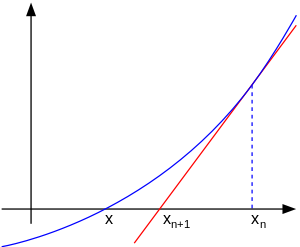


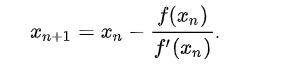
Рисунок 1. Иллюстрация метода Ньютона.

Иллюстрация метода Ньютона (синим изображена функция f(x), ноль которой необходимо найти, красным — касательная в точке очередного приближения xn. Здесь мы можем увидеть, что последующее приближение xn+1 лучше предыдущего xn.

Основная идея метода заключается в следующем: задаётся начальное приближение вблизи предположительного корня, после чего строится касательная к графику исследуемой функции в точке приближения, для которой находится пересечение с осью абсцисс. Эта точка берётся в качестве следующего приближения. И так далее, пока не будет достигнута необходимая точность.

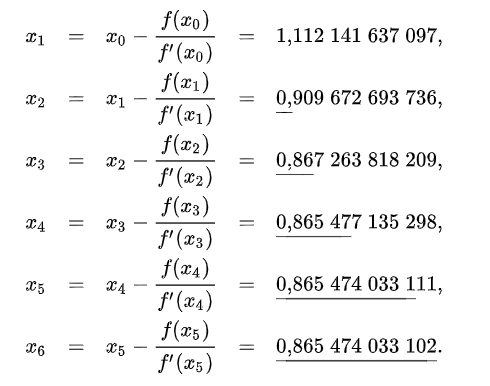
## Алгоритм

1. Задается начальное приближение x0.
2. Задается начальное приближение |xn+1 - xn| < eps или |f(xn+1) < eps| (то есть погрешность в нужных пределах), вычисляют новое приближение:



## Пример

Рассмотрим задачу о нахождении положительных x, для которых cos x=x^3. Эта задача может быть представлена как задача нахождения нуля функции f(x)=cos x-x^3. Имеем выражение для производной f'(x)=-sin x-3x^2. Так как cos x<=1 для всех x и x^3>1 для x>1, очевидно, что решение лежит между 0 и 1. Возьмём в качестве начального приближения значение x0=0,5, тогда:



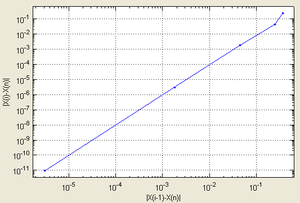
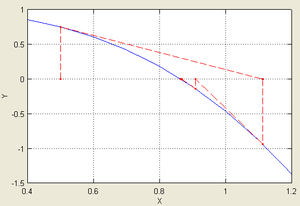


Рисунок 3. Иллюстрация применения метода Ньютона к функции f(x)=cos x-x^3 с начальным приближением в точке x0=0,5.

## Недостатки

При всех своих достоинствах метод Ньютона имеет ряд недостатков, которые ограничивают его применение. Основные из них:

1. Функция должна быть непрерывна и дифференцируема на всём исследуемом отрезке;
2. Производная функции должна существовать и быть непрерывна на всём исследуемом интервале:
3. Как все итерационные методы, метод Ньютона требует единственности решения на всём исследуемом интервале;
4. Если начальное приближение выбрано неправильно (слишком далеко от предполагаемого корня или меньше его), метод может не сойтись.

# Реализация и сравнение эффективности программной реализации алгоритма

Составим программу для поиска максимума и минимума на некотором интервале для следующей функции: cos(x) - (x \* x \* x)

Вначале продифференцируем эту функцию.

Производная первого порядка этой функции имеет вид:

-sin(x) - 3 \* (x \* x)

Вычисление производных при этом лучше вынести в отдельную функцию, чтобы упростить реализацию самого метода.

Ниже приведены примеры исходного кода поиска максимума. Если экстремум не является максимумом, будет возбуждено исключение.

Реализация на C++

#define \_USE\_MATH\_DEFINES

#include <time.h> // там clock(), clock\_t, CLOCKS\_PER\_SEC

#include <intrin.h> // там \_\_rdtsc()

#include <windows.h> // там функции использования HPET QueryPerformanceCounter(&tackts) и QueryPerformanceFrequency(&freq)

#include <math.h>

#include <iostream>

#include <string>

using namespace std;

#define SIZE\_DARR 100000

double arr[SIZE\_DARR];

double Dfdx(double x) {

return cos(x) - (x \* x \* x);

}

double D2fdx2(double x) {

return -sin(x) - 3 \* (x \* x);

}

void Newton(double x, double eps) { // заполняет массив синусами

double f, df;

do {

f = Dfdx(x);

df = D2fdx2(x);

x = x - f / df;

} while (fabs(f) > eps);

}

// частота в мегагерцах через RDTSC

int mhz\_cpu() {

clock\_t clock\_tick1, clock\_tick2;

\_\_int64 cpu\_tick1, cpu\_tick2;

int rdtsc\_tick;

double usec; // время в микросекундах

clock\_tick1 = clock();

while ((clock\_tick2 = clock()) == clock\_tick1); // пропуск остатка текущего тика clock

cpu\_tick1 = \_\_rdtsc(); // взять TSC

while (clock() == clock\_tick2); // отсчет одного тика clock

cpu\_tick2 = \_\_rdtsc() - cpu\_tick1; // сколько натикал счетчик TSC за один тик clock

// вычисляем частоту в мегагерцах

usec = 1000000.0 / CLOCKS\_PER\_SEC; // время одного тика clock в микросекундах

return int(cpu\_tick2 / usec);

}

int main()

{

int sz;

clock\_t clock\_start, clock\_time; // стартовое и измеренное время для clock()

\_\_int64 hpet\_start, hpet\_end, hpet\_time, // стартовое, финишное и измеренное время для HPET

hpet\_freq; // частота HPET

\_\_int64 rdtsc\_start, rdtsc\_time, // стартовое, финишное и измеренное время для RDTSC

rdtsc\_freq; // частота счетчика тактов (процессора) в мегагерцах

char ans; // ответ на запрос о продолжении

setlocale(LC\_CTYPE, "rus"); // кириллизация

cout << "Испытываем clock()\n\n";

sz = SIZE\_DARR; // задаем число инициируемых синусами элементов массива

int n, i;

do { // повторять, чтобы оценить повторяемость результатов

for (double eps = 0.1; eps >= 0.0000000001; eps \*= 0.1) {

clock\_start = clock(); // стартовая засечка

for (int i = 0; i < SIZE\_DARR; i++)

Newton(0.5, eps);

clock\_time = clock() - clock\_start; // замер продолжительности

cout << clock\_time << " мсек " << eps << "eps" << endl;

}

cout << "\nПовторить(1 - да)?: ";

cin >> ans; // ввод ответа на запрос

} while (ans == '1');

cout << "\nИспытываем RDTSC\n\n";

sz = SIZE\_DARR; // задаем число инициируемых синусами элементов массива

do { // повторять, чтобы оценить повторяемость результатов

for (double eps = 0.1; eps >= 0.0000000001; eps \*= 0.1) {

rdtsc\_start = \_\_rdtsc();

for (int i = 0; i < SIZE\_DARR; i++)

Newton(0.5, eps);

rdtsc\_time = \_\_rdtsc() - rdtsc\_start;

cout << rdtsc\_start << ": " << rdtsc\_time << " тактов";

rdtsc\_freq = mhz\_cpu();

cout << "\nВремя = " << int(0.001 \* rdtsc\_time / rdtsc\_freq) << " мсек " << eps << "eps";

}

cout << "\nПовторить(1 - да)?: ";

cin >> ans; // ввод ответа на запрос

} while (ans == '1');

printf("\nЗавершено");

return 0;

}

Реализация на C#

using System;

using System.Diagnostics;

using System.Diagnostics.CodeAnalysis;

namespace lab1.\_1

{

class Program

{

static double Dfdx(double x)

{

return Math.Cos(x) - (x \* x \* x);

}

static double D2fdx2(double x)

{

return -Math.Sin(x) - 3 \* (x \* x);

}

private static void Newton(double x, double eps)

{

double f, df;

do

{

f = Dfdx(x);

df = D2fdx2(x);

x = x - f / df;

} while (Math.Abs(f) > eps);

}

static void Main(string[] args)

{

Console.WriteLine("2. f(x) = cos(x) - (x^3) | x0 = 0.5");

Stopwatch sw = new Stopwatch();

for (double eps = 0.1; eps >= 0.0000000001; eps \*= 0.1)

{

sw.Start(); // стартовая засечка

for (int i = 0; i < 100000; i++)

Newton(0.5, eps);

sw.Stop(); // замер продолжительности

Console.WriteLine($"{sw.ElapsedMilliseconds} мс {eps} eps");

}

}

}

}

Реализация на JavaScript

// const { performance } = require('perf\_hooks');

function Dfdx(x) {

return Math.cos(x) - (x \* x \* x);

}

function D2fdx2(x) {

return -Math.sin(x) - 3 \* (x \* x);

}

function Newton(x, eps) {

let f, df;

do {

f = Dfdx(x);

df = D2fdx2(x);

x = x - f / df;

} while (Math.abs(f) > eps);

return x;

}

console.log('eps\t\tmilliseconds');

let times = [];

let eps = 0.1;

for (let i = 0; i < 9; i++) {

let time = performance.now(); // стартовая засечка

for (let i = 0; i < 100000; i++)

result = Newton(0.5, eps);

time = performance.now() - time;// замер продолжительности

times.push(time);

eps \*= 0.1;

}

let e = 0.1;

for (let time of times) {

console.log(`${e.toFixed(9)} \t ${time.toFixed(5)}`);

e \*= 0.1;

}

Для нахождения минимума условие определения экстремума изменяется на противоположное D2fdx2(x1) > 0.

Исследуем зависимость скорости выполнения программного кода от

1. а) заданной погрешности вычисления
2. б) языка программирования

f(x) = cos(x) - (x^3) | x0 = 0.5

|  |  |  |  |
| --- | --- | --- | --- |
| Погрешность вычисления, | Время выполнения программного кода, мс | | |
| C# | C++ | JavaScript |
| 0.01 | 24 | 55 | 53 |
| 0.001 | 49 | 46 | 40 |
| 0.0001 | 81 | 57 | 26 |
| 0.00001 | 124 | 58 | 20 |
| 0.000001 | 166 | 59 | 21 |
| 0.0000001 | 213 | 68 | 24 |
| 0.00000001 | 258 | 70 | 27 |
| 0.000000001 | 311 | 69 | 31 |
| 0.0000000001 | 362 | 72 | 42 |

Анализ охватывает следующие средства:

1. **Язык С++ - режим Debug, функция Clock(), использование команды RDTSC**

На всех UNIX-подобных ОС, очень старая функция clock( ) возвращает процессорное время процесса в тиках, а макрос CLOCKS\_PER\_SEC количество тиков в секунду.

Функция clock() описывается в time.h и возвращает число тиков от момента загрузки программы. Тик обычно равен 1 миллисекунде, но для возможности в последующем работать с другой длительностью такта в time.h фиксируется константа CLOCKS\_PER\_SEC (время изерения было взять в нс).

Счетчик тактов процессора TSC (Time Stamp Counter) встроен в ядро CPU и обычно прирастает на 1 с каждым тактом задающего генератора (это тот, что определяет гигагерцы процессора). В С/С++ он используется через функцию \_\_int64 \_\_ rdtsc(), описанную в windows.h.

1. **С# режим Debug, Stopwatch Класс**

Класс Stopwatch основан на HPET (High Precision Event Timer, таймер событий высокой точности). Данный таймер был введён фирмой Microsoft, чтобы раз и навсегда поставить точку в проблемах измерения времени. Частота этого таймера (минимум 10 МГц) не меняется во время работы системы. Для каждой системы Windows сама определяет, с помощью каких устройств реализовать этот таймер.

На многопроцессорном компьютере не имеет значения, на каком процессоре выполняется поток. Однако из-за ошибок в BIOS или слое абстрагирования оборудования (HAL) можно получить разные временные результаты на разных процессорах. Чтобы указать соответствие процессоров для потока, используйте ProcessThread.ProcessorAffinity метод.

1. **JavaScript, console.time() и performance.now()**

В High Resolution Time API есть функция now(), возвращающая объект DOMHighResTimeStamp. Это число с плавающей точкой, отражающее текущее время в миллисекундах, с точностью до тысячной миллисекунды.

## Выводы:

По результатам экспериментов можно сделать вывод, что скорость работы программ на языке C++ выше, чем скорость работы программы, реализующий метод Ньютона на языке C#. Это связано с тем, что С++ компилируется непосредственно в машинный код и работает с минимально возможным количеством хелперов и прослоек. Шаблоны C# определяются во время выполнения и это медленнее, чем шаблоны времени компиляции C++. С уменьшение погрешности вычислений, скорость выполнения программ не растет, что связано с тем, что метода касательных имеет квадратичную сходимость, т.е малые значения погрешности достигаются даже при небольшом количестве итераций, т.е. уменьшение погрешности не приводит к увеличению числа итераций.

Замеры времени на языках С++ и С# проводились в режиме Debug. В конфигурации Release время исполнения пустой функции Newton равно 0. Это объясняется тем, что в конфигурации Release удаляется отладочная информация из исполняемого файла. Это приводит к уменьшению размера исполняемого файла (обычно в несколько раз). Исключаются дополнительные проверки, например, инициализированы переменные или нет. Поэтому в конфигурации Release программа может работать значительно быстрее.

С JavaScript могут происходить разные события такие как:

1. **Случайное измерение ненужных вещей**

Скорость отправки данных и вывода в консоль сильно зависит от браузера и даже от того, что ещё он делает в это время. Вероятно, вы считаете, что это console.log работает непредсказуемо медленно. Но в любом случае будет ошибкой выполнять более одной функции, даже если каждая из функций не подразумевает никаких операций ввода-вывода.

1. **Однократное измерение**

Риск данного подхода заключается в том, что браузерный JavaScript-движок может выполнять субоптимизацию, т. е. во второй раз функция будет вызвана с теми же входными данными, которые будут запомнены и использованы в дальнейшем. Чтобы это обойти, можно использовать много разных входных строк вместо того, чтобы раз за разом брать одно и то же значение. Однако при разных входных данных и скорость выполнения функции раз от раза может отличаться.

1. **Излишнее доверие к средним значениям**

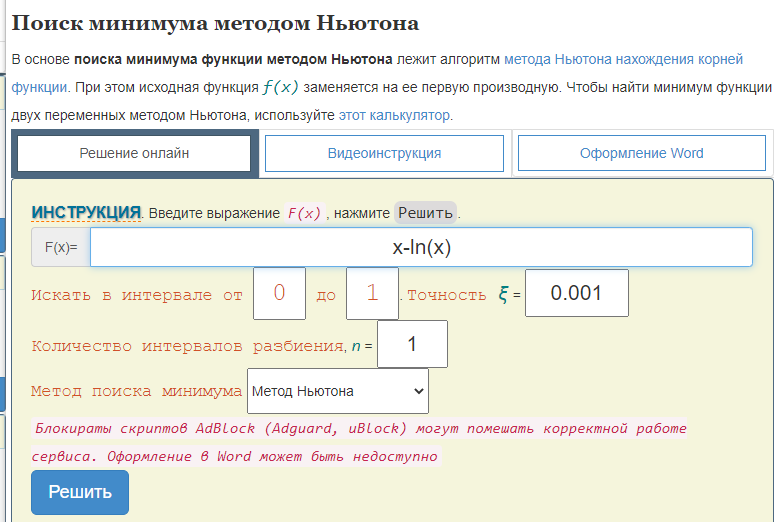
Скорее всего, причина как раз в проведении субоптимизации и в необходимости «прогрева» компилятора. Мало что можно сделать, чтобы этого избежать, но зато можно обезопасить себе от неверных заключений.

Пытаясь продемонстрировать точность измерения производительности в JavaScript с помощью performance.now(), обнаружилось, что наша интуиция может подвести нас: эмпирические данные совершенно не совпали с нашими предположениями.

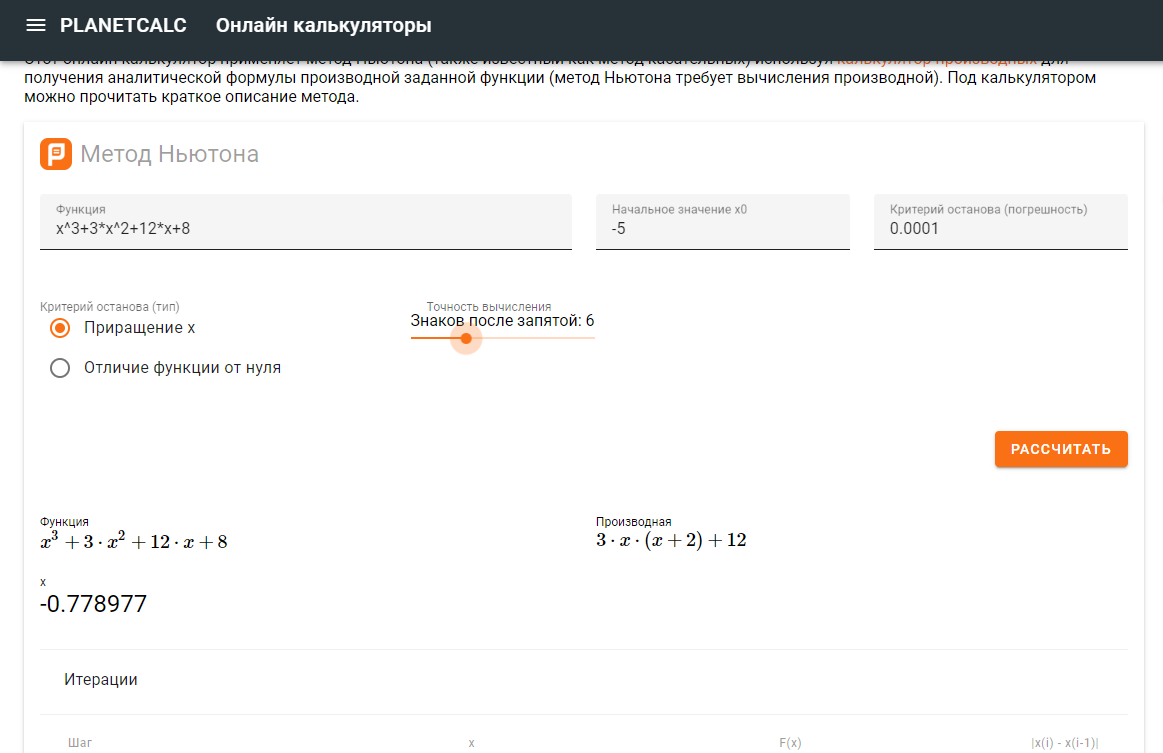
Рассмотренные ошибки — не единственные возможные. К ним можно добавить, например, измерение нереалистичных сценариев или измерение только на одном JS-движке.

# Онлайн реализации метода касательных

На сайте https://math.semestr.ru реализован поиск минимума методом Ньютона. Для нахождения минимума функции одной переменной методом касательных необходимо перейти по ссылке: https://math.semestr.ru/optim/minimum.php, выбрать в списке метод Ньютона, задать функцию, задать интервал, задать точность вычисления и нажать кнопку РЕШИТЬ.



На сайте https://planetcalc.ru/7748/ реализовано онлайн решение нелинейного уравнения методом касательных. Можно воспользоваться данным сервисом, для нахождения критических точек, в которых возможно достижение экстремума. Для этого в поле функция необходимо записать выражение для первой производной функции.



# Исследование реализации выбранного алгоритма в различных пакетах прикладных программ (MathCad; MATLAB).

Средства пакета MathCAD позволяют решить нелинейное уравнение вида f(x)=0 методом касательных. Т.е можно найти точки, в которых предположительно функция принимает оптимальное значение. Для решения уравнений MathCAD имеет встроенную функцию root, которая, в зависимости от типа задачи, может включать либо два, либо четыре аргумента и, соответственно, работает по-разному.

- root (f(х), х);

- root(f(х), х, а, b),

где f(х) – скалярная функция, определяющая уравнение;

х – скалярная переменная, относительно которой решается уравнение;

а, b – границы интервала, внутри которого происходит поиск корня.

Первый тип функции root требует дополнительного задания начального значения (guess value) переменной х. Для этого нужно предварительно присвоить этой переменной некоторое число, в окрестности которого будет производиться поиск корня. Таким образом, присвоение начального значения требует информации о примерной локализации корня. Отделить корень можно, построив график функции f(х) и с помощью опции Trace определить примерно абсциссу пересечения графиком оси ОХ.

**Пример.** Рассмотрим уравнение sin(x)=0, корни которого известны заранее. Примем начальное значение x=0,5.

**Решение.**

Реализуем метод Ньютона для оптимизации функции в Matlab.

Задача. Методом Ньютона найти точку минимума x\* и минимальное значение f\* функции f(x)=(x-2)4-lnx на отрезке xє[2;3] c точностью 10-7

Начнем с того, что создадим новый скрипт и назовем его Newton.m. Затем пропишем в нем код:

function [Xk, Yk] = Newton(f,diap)

a = diap(1); % границы

b = diap(2);

df = char(diff(sym(f))); % символьно ищем первую

ddf = char(diff(sym(df))); % и вторую производные

F = inline(f); % преобразуем в функции

F1 = inline(df);

F2 = inline(ddf);

eps = 0.0000001; % задаем точность

if F(a)\*F2(a) > 0 % проверка с какой границы начинать искать

Xk = b;

else

Xk = a;

end

while abs(F1(Xk)) > eps

X0 = Xk; % X0 - значение предыдущего шага

Xk = X0 - (F1(X0)/(F2(X0))); % расчет нового значения

Yk = F(Xk);

end

end

Вполне понятная функция, которая работает с первой и второй производной символьной функции, которую получает в качестве параметра. Также в параметрах принимается диапазон. Эта функция возвращает координаты точки и значение экстремума.

Теперь нам осталось вызвать эту функцию в командном окне:

>> fun = '(x-2)^4 - log(x)';

>> diap = [2,3];

>> [Xk, Yk] = Newton(fun, diap);

В итоге получилось:

Xk = 2.4663

Yk = -0.8554

Этот метод позволяет найти только локальный экстремум, и если на выбранном диапазоне есть несколько экстремумов, то метод может найти не тот, который нужен вам. Также, очень важно задавать как можно узкий диапазон поиска, иначе метод может работать некорректно, особенно это проявляется с периодическими функциями по типу cos(x) и т.п.

# Список источников:

1. Аттетков А.В., Галкин С.В., Зарубин B.C. МЕТОДЫ ОПТИМИЗАЦИИ Учеб. для вузов /Под ред. B.C. Зарубина, А. П. Крищенко. - 2-е изд., стереотип. - М.: МГТУ им. Н. Э. Баумана, 2003.
2. Маркина М.B., Судакова А.В. ПРАКТИКУМ ПО РЕШЕНИЮ ЗАДАЧ ОПТИМИЗАЦИИ В ПАКЕТЕ MATLAB: учебно-методическое пособие. –Нижний Новгород: Нижегородский госуниверситет, 2017.
3. Прокопенко Н. Ю. Методы оптимизации: учеб. пособие /Н. Ю. Прокопенко; Нижегор. гос. архитектур. - строит. ун-т. – Н. Новгород: ННГАСУ, 2018.