|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** | |

ФАКУЛЬТЕТ Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**Лабораторная работа № 6**

«Решение систем нелинейных уравнений методом Ньютона»

по курсу «Численные методы»

Выполнил:

студент группы ИУ9-62Б

Марченко Андрей

Проверила:

Домрачева А. Б.

Москва, 2024

1. **Цель**

1.1 Изучение и сравнение метода деления отрезка пополам и метода Ньютона (касательных) решения нелинейных уравнений.

1.2 Изучение метода Ньютона решения системы нелинейных уравнений.

1. **Постановка задачи**

2.1 **Дано:** Уравнение , имеющее на отрезке один корень

**Задание:**

* Нарисовать график функции и найти отрезки, где функция имеет простые корни и отличных от нуля первые две производные.
* С точностью найти все корни уравнения методом деления отрезка пополам и методом Ньютона, определить число приближений в каждом случае.
* Сравнить полученные результаты.

2.2

**Дано:** Система нелинейных уравнений

В векторной форме:

**Задание:**

* Решить данную систему графически и принять полученное решение за начальное приближения
* Решить систему методом Ньютона с точностью ;

1. **Теоретические сведения**

2.1

Метод деления отрезка пополам является самым простым методом решения заданного уравнения. Поиск корня уравнения начинаем с деления пополам отрезка – точкой . Из двух получившихся отрезков выбираем тот, на концах которого функция имеет разные знаки, то есть на котором содержится корень. Далее повторяем алгоритм для выбранного отрезка. Процесс повторяем пока не будет выполнено условие:

, где – требуемая точность

Метод Ньютона (касательных) решения нелинейного уравнения является итерационным. Для получения -й итерации (приближения) метода из точки , лежащей на графике функции, проводим касательную. Точка пересечения касательной с осью абсцисс и есть следующее приближение к корню уравнения. Алгебраически это можно записать так:

Заданная погрешность считается достигнутой, если выполняется условие:

2.2

Выбрав начальное приближение к решению системы, необходимо строить последующие приближения итерационно по следующей формуле:

При этом вычисляется как решение системы линейных уравнений с матрицей, равной матрице Якоби отображения в точке последнего приближения:

Чтобы достичь заданной точности необходимо проверять близость последних приближений по следующей норме

1. **Индивидуальный вариант**

Вариант 6:

4.1

4.2

1. **Реализация**

4.1

Листинг 1. Main.cpp

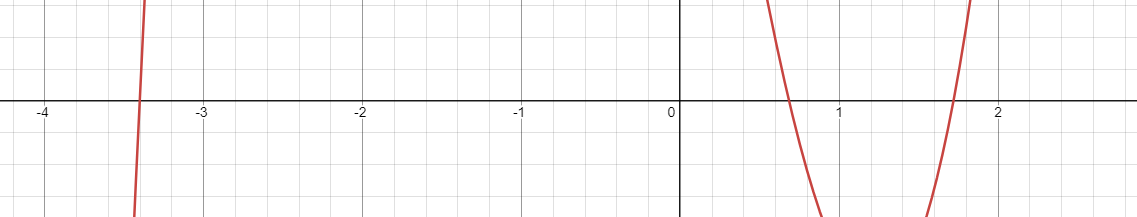
#include <iostream>  
  
double func(double x) {  
 return x \* x \* x + x \* x - 7 \* x + 4;  
}  
  
double func\_dx(double x) {  
 return 3 \* x \* x + 2 \* x - 7;  
}  
  
double func\_dxdx(double x) {  
 return 6 \* x + 2;  
}  
  
double findRoot(double a, double b, double epsilon) {  
 double c;  
 while ((b - a) > epsilon) {  
 c = (a + b) / 2;  
 if (func(c) == 0.0) {  
 return c;  
 } else if (func(a) \* func(c) < 0) {  
 b = c;  
 } else {  
 a = c;  
 }  
 }  
 return (a + b) / 2;  
}  
  
int sgn(double x) {  
 if (x > 0) return 1;  
 if (x == 0) return 0;  
 return -1;  
}  
  
double newtonMethod(double a, double b, double eps) {  
 double x\_prev, x\_cur;  
 if (func(a) \* func\_dxdx(a) > 0) {  
 x\_prev = a;  
 }  
 else {  
 x\_prev = b;  
 }  
 x\_cur = x\_prev - func(x\_prev) / func\_dx(x\_prev);  
 while (func(x\_cur) \* func(x\_cur + sgn(x\_cur - x\_prev) \* eps) > 0) {  
 double t = x\_prev - func(x\_prev) / func\_dx(x\_prev);  
 x\_prev = x\_cur;  
 x\_cur = t;  
 }  
  
 return x\_cur;  
}  
  
int main() {  
 std::cout << findRoot(-8, 0, 0.01) << std::endl;  
 std::cout << newtonMethod(-8, 0, 0.01) << std::endl;  
 return 0;  
}

4.2

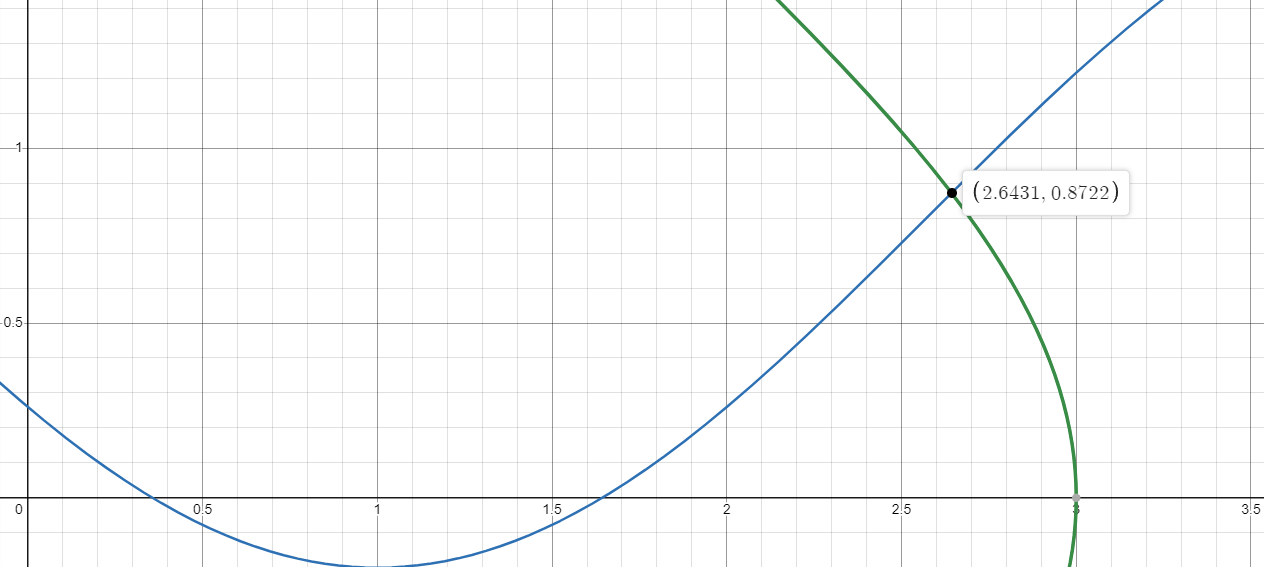
Листинг 2. Main.py

import numpy as np  
  
  
def newtonMethod(func, J, x0, eps=0.01):  
 x = x0  
 for \_ in range(100):  
 y = np.linalg.solve(J(x), -func(x))  
 x\_new = x + y  
 if np.max(np.abs(x\_new - x)) < eps:  
 return x\_new  
 x = x\_new  
  
 print("Алгоритм не сошелся за 100 итераций")  
 return x  
  
  
def func(x):  
 return np.array([  
 np.cos(x[0] - 1) + x[1] - 0.8,  
 x[0] - np.cos(x[1]) - 2  
 ])  
  
  
def J(x):  
 return np.array([  
 [-np.sin(x[0] - 1), 1],  
 [1, np.sin(x[1])]  
 ])  
  
  
x0 = np.array([3.0, 1.0])  
solution = newtonMethod(func, J, x0)  
print(solution)

1. **Результаты**

4.1 

| Отрезок поиска | Метод деления пополам | Метод Ньютона |
| --- | --- | --- |
|  | 7 | 2 |
|  | 7 | 2 |
|  | 8 | 6 |
| Количество приближений при поиске корня различными методами | | |

4.2 

Результаты: [2.64310698 0.87224765]

1. **Выводы**

4.1

В ходе выполнения лабораторной работы были реализованы методы приближенного решения нелинейного уравнения: метод деления отрезка пополам и метод Ньютона. Сравнение результатов запуска программы показывает, что метод Ньютона сходится гораздо быстрее. При этом его дополнительной сложностью является вычисление производной функции для построения касательных.

4.2

В ходе выполнения лабораторной работы был изучен и реализован метод

Ньютона приближенного решения системы нелинейных уравнений от двух переменных.

Важной чертой метода является то, что для его применения необходимо вычисление Якобиана векторной функции. При этом, недостатком является условие достаточной близости начального приближения к ответу.

Однако, метод работает достаточно быстро и получает решение с нужной точностью и из достаточно произвольных точек плоскости.