**Московский авиационный институт**

**(национальный исследовательский университет)**

Факультет: «Компьютерные науки и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа №2**

по курсу «Численные методы»

Тема: «Решение нелинейных уравнений и систем нелинейных уравнений»

Студент: Мариничев И. А.

Группа: М8О-308Б-19

Преподаватель: Пивоваров Д. Е.

Оценка:

Москва

2022

**1. Постановка задачи.**

**Вариант №12**.

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.



2.2. Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

Значение параметра : 3



**2. Описание методов.**

**2.1. Метод простых итераций.**

Решаемая задача: Решение нелинейных уравнений и систем нелинейных уравнений.

Тип метода: Итерационный метод.

Основная идея: Выразить неизвестную переменную из исходного уравнения так, чтобы q было строго меньше 1.

Сложность: O () на каждой итерации.

Преимущества: Простота реализации, отсутствие необходимости знаний о первой производной функции.

**2.2. Метод Ньютона.**

Решаемая задача: Решение нелинейных уравнений и систем нелинейных уравнений.

Тип метода: Итерационный метод.

Основная идея: Движемся по касательной на каждой итерации.

Сложность: O ().

Преимущества: Находит решение задачи сильно быстрее, чем МПИ, если верно выделена область поиска.

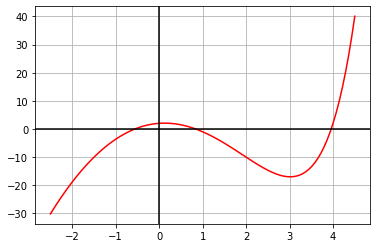
**3. Демонстрация работы программы.**

**3.1. Анализ функции**

Посмотрим на график функции

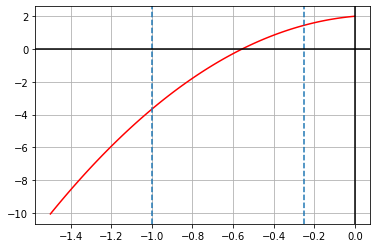


import matplotlib.pyplot as plt  
from scipy.misc import derivative  
import numpy as np  
  
x = np.linspace(-2.5, 4.5, 100)  
f = pow(3, x) - 5 \* x\*\*2 + 1 *# вариант 12*  
  
plt.plot(x, f,color="red")  
plt.grid(True, which='both')  
plt.axhline(y=0, color='k')  
plt.axvline(x=0, color='k')  
plt.show()



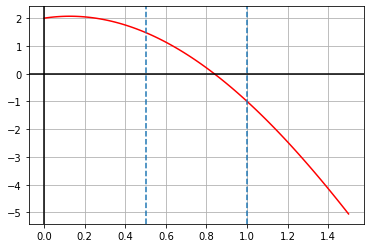
Посмотрим на первое решение. Возьмем отрезок [-1, -0.25] для его поиска

a1, b1 = -1, -0.25  
  
x = np.linspace(-1.5, 0, 100)  
f = pow(3, x) - 5 \* x\*\*2 + 1  
  
plt.plot(x, f,color="red")  
plt.axvline(a1, linestyle='--')  
plt.axvline(b1, linestyle='--')  
plt.grid(True, which='both')  
plt.axhline(y=0, color='k')  
plt.axvline(x=0, color='k')  
plt.show()



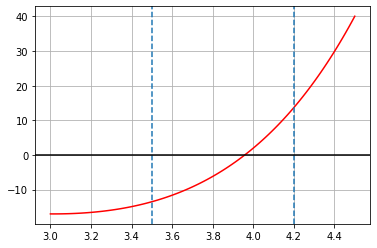
Посмотрим на второе решение. Возьмем отрезок [0.5, 1] для его поиска

a2, b2 = 0.5, 1  
  
x = np.linspace(0, 1.5, 100)  
f = pow(3, x) - 5 \* x\*\*2 + 1  
  
plt.plot(x, f,color="red")  
plt.axvline(a2, linestyle='--')  
plt.axvline(b2, linestyle='--')  
plt.grid(True, which='both')  
plt.axhline(y=0, color='k')  
plt.axvline(x=0, color='k')  
plt.show()



Посмотрим на третье решение. Возьмем отрезок [3.5, 4.2] для его поиска

a3, b3 = 3.5, 4.2  
  
x = np.linspace(3, 4.5, 100)  
f = pow(3, x) - 5 \* x\*\*2 + 1  
  
plt.plot(x, f,color="red")  
plt.axvline(a3, linestyle='--')  
plt.axvline(b3, linestyle='--')  
plt.grid(True, which='both')  
plt.axhline(y=0, color='k')  
plt.show()

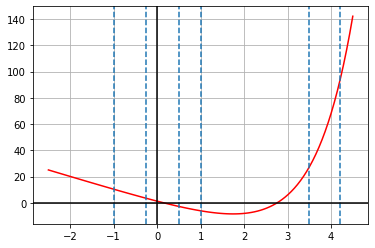


**Первая производная:**



Для нас важно, чтобы на выбранных выше отрезках знак производной не менялся. Убедимся в этом

**def** function(x):  
 **return** pow(3, x) - 5 \* x\*\*2 + 1  
  
**def** function\_diff(x):  
 **return** derivative(function, x)  
  
x = np.linspace(-2.5, 4.5, 100)  
  
plt.plot(x, function\_diff(x), color="red")  
plt.axvline(a1, linestyle='--')  
plt.axvline(b1, linestyle='--')  
plt.axvline(a2, linestyle='--')  
plt.axvline(b2, linestyle='--')  
plt.axvline(a3, linestyle='--')  
plt.axvline(b3, linestyle='--')  
plt.grid(True, which='both')  
plt.axhline(y=0, color='k')  
plt.axvline(x=0, color='k')  
plt.show()



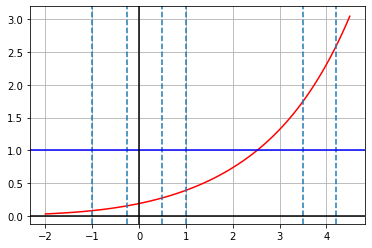
**Производная функции**



Для нас важно, чтобы q =



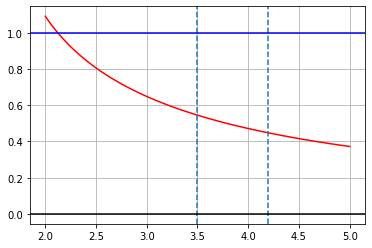
**def** phi(x):  
 **return** np.sqrt((pow(3, x) + 1) / 5)  
  
**def** phi\_diff(x):  
 **return** derivative(phi, x)  
  
x = np.linspace(-2, 4.5, 100)  
  
plt.plot(x, phi\_diff(x), color="red")  
plt.axvline(a1, linestyle='--')  
plt.axvline(b1, linestyle='--')  
plt.axvline(a2, linestyle='--')  
plt.axvline(b2, linestyle='--')  
plt.axvline(a3, linestyle='--')  
plt.axvline(b3, linestyle='--')  
plt.grid(True, which='both')  
plt.axhline(y=0, color='k')  
plt.axhline(y=1, color='b')  
plt.axvline(x=0, color='k')  
plt.show()



Как можно заметить нам подхдят только два первых отрезка. Тогда попробуем выразить иначе



**def** log(a, b):  
 **return** np.log(b) / np.log(a)  
  
**def** phi(x):  
 **return** log(3, 5 \* x\*\*2 - 1)  
  
**def** phi\_diff(x):  
 **return** derivative(phi, x)  
  
x = np.linspace(2, 5, 100)  
  
plt.plot(x, phi\_diff(x), color="red")  
plt.axvline(a3, linestyle='--')  
plt.axvline(b3, linestyle='--')  
plt.grid(True, which='both')  
plt.axhline(y=0, color='k')  
plt.axhline(y=1, color='b')  
plt.show()



Теперь условие выполняется для третьего отрезка. Соответсвенно, для первых двух отрезков будем использовать первый вариант выражения , а для третьего второй вариант

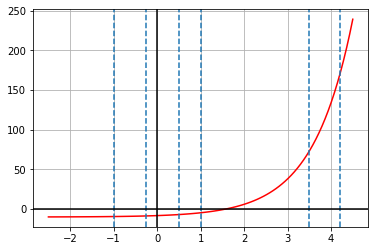


**Вторая производная:**



Для нас важно, чтобы на выбранных выше отрезках знак второй производной не менялся. Убедимся в этом

**def** function\_second\_diff(x):  
 **return** derivative(function\_diff, x)  
  
x = np.linspace(-2.5, 4.5, 100)  
  
plt.plot(x, function\_second\_diff(x), color="red")  
plt.axvline(a1, linestyle='--')  
plt.axvline(b1, linestyle='--')  
plt.axvline(a2, linestyle='--')  
plt.axvline(b2, linestyle='--')  
plt.axvline(a3, linestyle='--')  
plt.axvline(b3, linestyle='--')  
plt.grid(True, which='both')  
plt.axhline(y=0, color='k')  
plt.axvline(x=0, color='k')  
plt.show()



**Численное решение**

Теперь запустите в терминале программу на **С++** при помощи команд

make  
make run

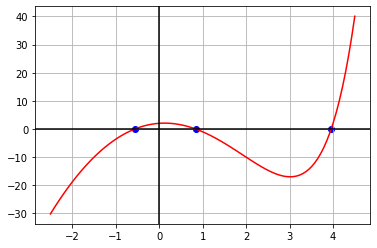
В трех появившихся файлах answer\_NN.txt можно увидеть полученные численные решения. Проверим их

x\_1 = -0.555548  
x\_2 = 0.837941  
x\_3 = 3.95759  
  
print("Значение функции f(x) в найденных точках:")  
print(f"1) f({x\_1}) = {function(x\_1):0.2f}")  
print(f"2) f({x\_2}) = {function(x\_2):0.2f}")  
print(f"3) f({x\_3}) = {function(x\_3):0.2f}")

Значение функции f(x) в найденных точках:  
1) f(-0.555548) = 0.00  
2) f(0.837941) = 0.00  
3) f(3.95759) = 0.00

Наконец, построим полученные точки на графике

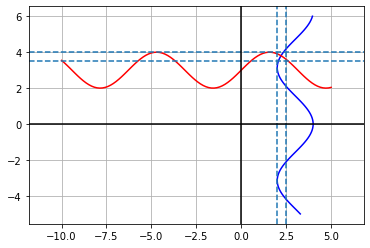
x\_0 =[x\_1, x\_2, x\_3]  
   
y\_0 =[function(x\_1), function(x\_2), function(x\_3)]  
  
x = np.linspace(-2.5, 4.5, 100)  
f = pow(3, x) - 5 \* x\*\*2 + 1  
  
plt.plot(x, f,color="red")  
plt.grid(True, which='both')  
plt.axhline(y=0, color='k')  
plt.axvline(x=0, color='k')  
plt.scatter(x\_0, y\_0, c ="blue")  
plt.show()



**3.2. Анализ системы**

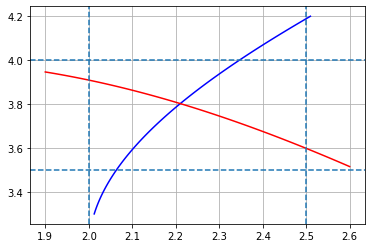
Посмотрим на график системы:

import matplotlib.pyplot as plt  
import numpy as np  
  
X1 = np.linspace(-10,5,100)  
X2 = np.linspace(-5, 6, 100)  
  
*# вариант 12*  
x1 = np.cos(X2) + 3   
x2 = np.sin(X1) + 3  
  
*# область G*  
a, b = 2, 3.5  
c, d = 2.5, 4  
  
plt.plot(x1, X2, color="blue")  
plt.plot(X1, x2, color="red")  
plt.axvline(a, linestyle='--')  
plt.axhline(b, linestyle='--')  
plt.axvline(c, linestyle='--')  
plt.axhline(d, linestyle='--')  
plt.grid(True, which='both')  
plt.axhline(y = 0, color='k')  
plt.axvline(x = 0, color='k')  
plt.axis('equal')  
plt.show()



Посмотрим на решение. Возьмем область для его поиска

X1 = np.linspace(1.9, 2.6,100)  
X2 = np.linspace(3.3, 4.2, 100)  
  
x1 = np.cos(X2) + 3   
x2 = np.sin(X1) + 3   
  
plt.plot(x1, X2, color="blue")  
plt.plot(X1, x2, color="red")  
plt.axvline(a, linestyle='--')  
plt.axhline(b, linestyle='--')  
plt.axvline(c, linestyle='--')  
plt.axhline(d, linestyle='--')  
plt.grid(True, which='both')  
plt.show()



**Численное решение**

Теперь запустите в терминале программу на **С++** при помощи команд

make  
make run

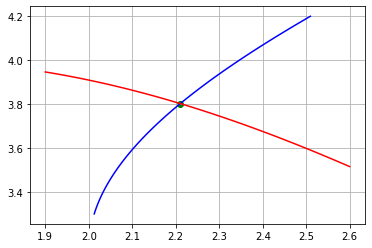
В трех появившихся файлах answer\_NN.txt можно увидеть полученное численное решение c разной точностью. Проверим его

**def** f1(x1, x2):  
 **return** x1 - np.cos(x2) - 3  
  
**def** f2(x1, x2):  
 **return** x2 - np.sin(x1) - 3  
  
x1\_0 = 2.21045  
x2\_0 = 3.80231  
  
print("Значение функций f1(x1, x2) и f2(x1, x2) в найденной точке:")  
print(f"f1{x1\_0, x2\_0} = {f1(x1\_0, x2\_0):0.2f}")  
print(f"f2{x1\_0, x2\_0} = {f2(x1\_0, x2\_0):0.2f}")

Значение функций f1(x1, x2) и f2(x1, x2) в найденной точке:  
f1(2.21045, 3.80231) = 0.00  
f2(2.21045, 3.80231) = 0.00

Наконец, построим полученную точку на графике

X1 = np.linspace(1.9, 2.6,100)  
X2 = np.linspace(3.3, 4.2, 100)  
  
x1 = np.cos(X2) + 3   
x2 = np.sin(X1) + 3   
  
plt.plot(x1, X2, color="blue")  
plt.plot(X1, x2, color="red")  
plt.grid(True, which='both')  
plt.scatter(x1\_0, x2\_0, c ="green")  
plt.show()



**4. Выводы.**

В ходе данной лабораторной работы я изучил базовые численных методы, решающие задачи нелинейной алгебры. При помощи метода простых итераций и метода Ньютона удалось найти решение нелинейного уравнения и системы двух нелинейных уравнений. Основная особенность этих методов состоит в том, что перед тем, как начинать искать решение необходимо провести анализ тех функций, с которыми мы работаем и выделить корректные области для поиска нужного решения.

Стоит также отметить, что те варианты реализаций, которые были написаны мной на С++ носят скорее учебный характер, так как передо мной стояла задача понять алгоритмы при реализации, а не написать максимально оптимальные решатели тех или иных задач.