Федеральное государственное автономное образовательное учреждение высшего образования Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №2

Вариант № 1

Выполнил: студент группы P3208, Васильев Н. А.

Преподаватель: Машина Е.А.

Санкт-Петербург 2025

# Текст задания

Численное решение нелинейных уравнений и систем.

Цель работы

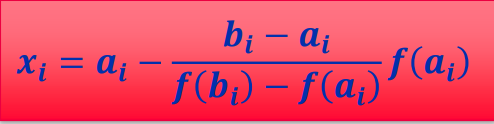
Изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов

Описание метода, расчётные формулы

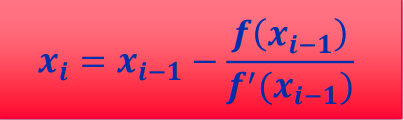
Метод половинного деления:

Метод секущих:

Метод хорд:



Метод Ньютона:

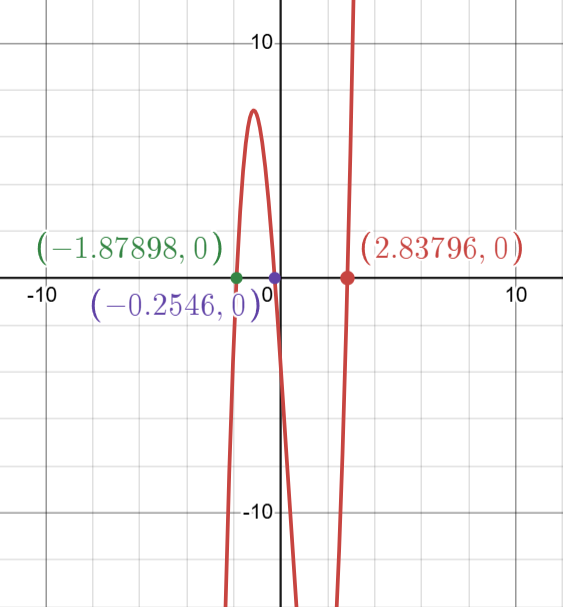


Метод секущих:

Вычислительная реализация задачи:

1 часть. Решение нелинейного уравнения

1. Отделить корни заданного нелинейного уравнения графически.



1. Определить интервалы изоляции корней.

Определение интервалов изоляции корней нелинейного уравнения — это процесс нахождения отрезков, на которых содержится ровно один корень уравнения.

Найдем точки пересечения графика с осью : .

Найдем интервалы изоляции корней:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Интервалы изоляции корней: , и .

1. Уточнить корни нелинейного уравнения с точностью .
2. Используемые методы для уточнения каждого из 3-х корней многочлена.

Метод половинного деления – крайний правый корень.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № шага |  |  |  |  |  |  |  |
| 1 | 2,750 | 3,000 | 2,875 | -3,350 | 7,050 | 1,510 | 0,250 |
| 2 | 2,750 | 2,875 | 2,813 | -3,350 | 1,510 | -0,985 | 0,125 |
| 3 | 2,813 | 2,875 | 2,844 | -0.985 | 1,510 | 0,242 | 0,062 |
| 4 | 2,813 | 2,844 | 2,829 | -0.985 | 0,242 | -0,376 | 0,031 |
| 5 | 2,829 | 2,844 | 2,837 | -0,376 | 0,242 | -0,058 | 0,016 |
| 6 | 2,837 | 2,844 | **2,841** | -0.058 | 0,242 | 0,101 | 0,008 |

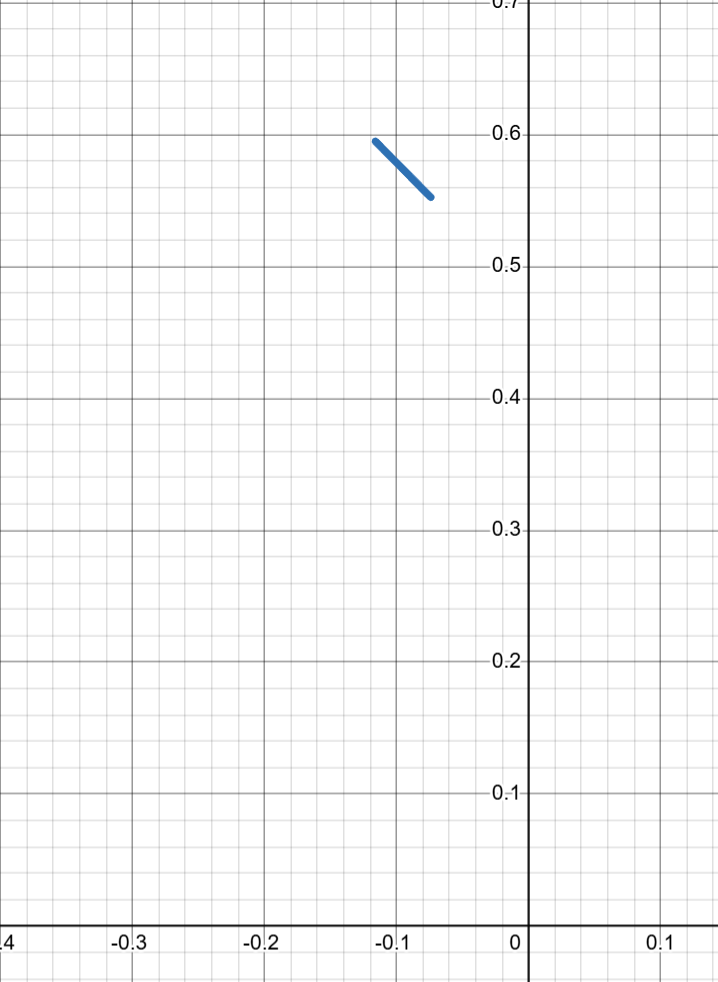
Метод секущих – крайний левый корень.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № шага |  |  |  |  |  |
| 1 | 2,750 | 3,000 | 2,831 | -0,295 | 0,169 |
| 2 | 3,000 | 2,831 | **2,837** | -0,024 | 0,007 |

Метод простой итерации – центральный корень.

Найдем производную функции

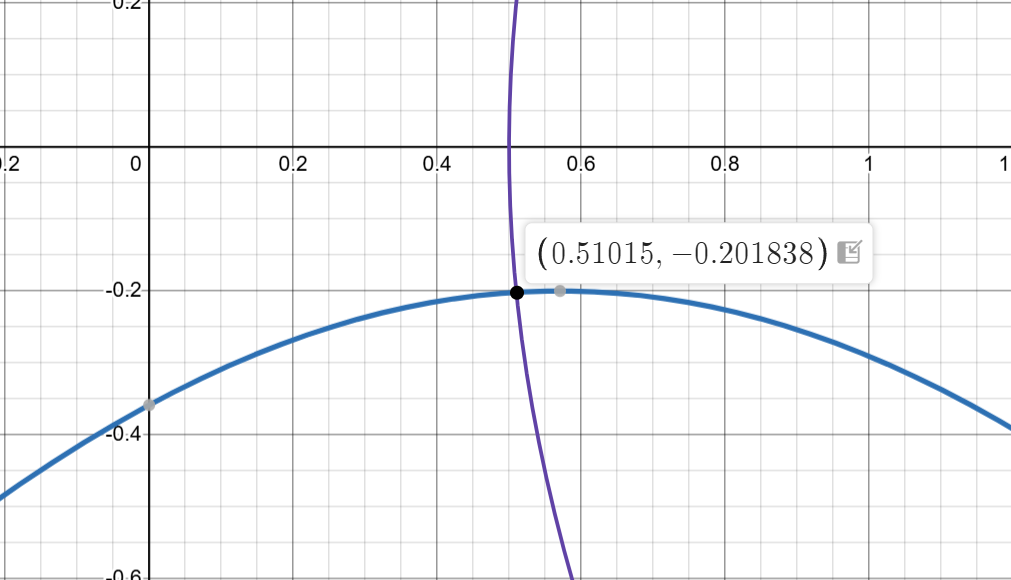
На границах интервала производная равна:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № шага |  |  |  |  |
| 1 | -0,500 | -0,276 | 0,293 | 0,224 |
| 2 | -0,276 | -0,256 | 0,019 | 0,020 |
| 3 | -0,256 | **-0,255** | 0,0001 | 0,001 |

2 часть. Решение системы нелинейных уравнений

В привычном виде получаем:



Области изоляции корней:

Проверим выполнение условия сходимости:

Следовательно, условие сходимости на области G выполнено, и метод простых итераций будет сходиться.

Возьмем начальное приближение в центре G:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № итерации |  |  |  |  |  |  |
| 1 | 0,500 | 0,561 | 0,061 | -0,500 | -0,202 | 0,297 |
| 2 | 0,561 | 0,510 | 0,051 | -0,202 | -0,200 | 0,002 |
| 3 | 0,510 | **0,510** | 0,0001 | -0,200 | **-0,202** | 0,002 |

Листинг программы

def method\_of\_chords(f, a, b, epsilon, max\_iter=1000):  
 if not verificator.check\_root(f, a, b):  
 return None  
  
 x0, x1 = a, b  
 for \_ in range(max\_iter):  
 f0, f1 = f(x0), f(x1)  
 if abs(f1 - f0) < 1e-12:  
 break  
 x2 = x1 - f1 \* (x1 - x0) / (f1 - f0)  
 if abs(x2 - x1) < epsilon:  
 plots.plot\_function(f, a, b)  
 return x2, f(x2), \_ + 1  
 x0, x1 = x1, x2  
 plots.plot\_function(f, a, b)  
 return max\_iter  
  
  
def newtons\_method(f, df, x0, epsilon, max\_iter=1000):  
 for \_ in range(max\_iter):  
 f0, df0 = f(x0), df(f, x0)  
 if abs(df0) < 1e-12:  
 break  
 x1 = x0 - f0 / df0  
 if abs(x1 - x0) < epsilon:  
 plots.plot\_function\_without\_points(f, x0)  
 return x1, f(x1), \_ + 1  
 x0 = x1  
 plots.plot\_function\_without\_points(f, x0)  
 return max\_iter  
  
  
def simple\_iterations(f, phi, x0, epsilon, max\_iter=1000):  
 x = x0  
 for \_ in range(max\_iter):  
 x\_next = phi(f, x)  
 if abs(x\_next - x) < epsilon:  
 plots.plot\_function\_without\_points(f, x0)  
 return x\_next, f(x\_next), \_ + 1  
 x = x\_next  
 plots.plot\_function\_without\_points(f, x0)  
 return max\_iter  
  
  
def newton\_system\_method(f, J, x0, epsilon, max\_iter=1000):  
 x = np.array(x0)  
 errors = []  
 for \_ in range(max\_iter):  
 F = np.array([fi(x) for fi in f])  
 J\_matrix = J(f, x)  
 try:  
 J\_inv = np.linalg.inv(J\_matrix)  
 except np.linalg.LinAlgError:  
 return None  
 delta = np.dot(J\_inv, F)  
 x\_next = x - delta  
 error = np.linalg.norm(x\_next - x)  
 errors.append(error)  
 if error < epsilon:  
 plots.plot\_functions\_system(f[0], f[1], x0)  
 return x\_next, \_, errors  
 x = x\_next  
 return max\_iter

Примеры и результаты работы программы

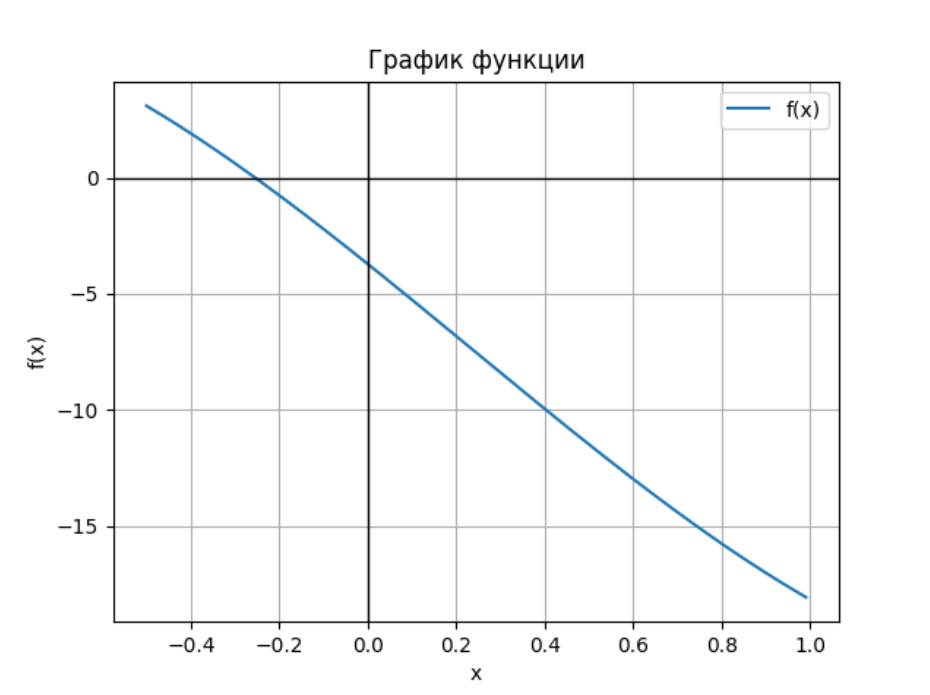
**Пример №1**. Решение уравнения 2.74 \* x\*\*3 - 1.93 \* x\*\*2 - 15.28 \* x - 3.72 методом хорд с точностью 0,01, границы [-0.5; 1]:

**Вывод:**

Корень: -0.2545898431759913

Значение функции: -0.00017614244687402802

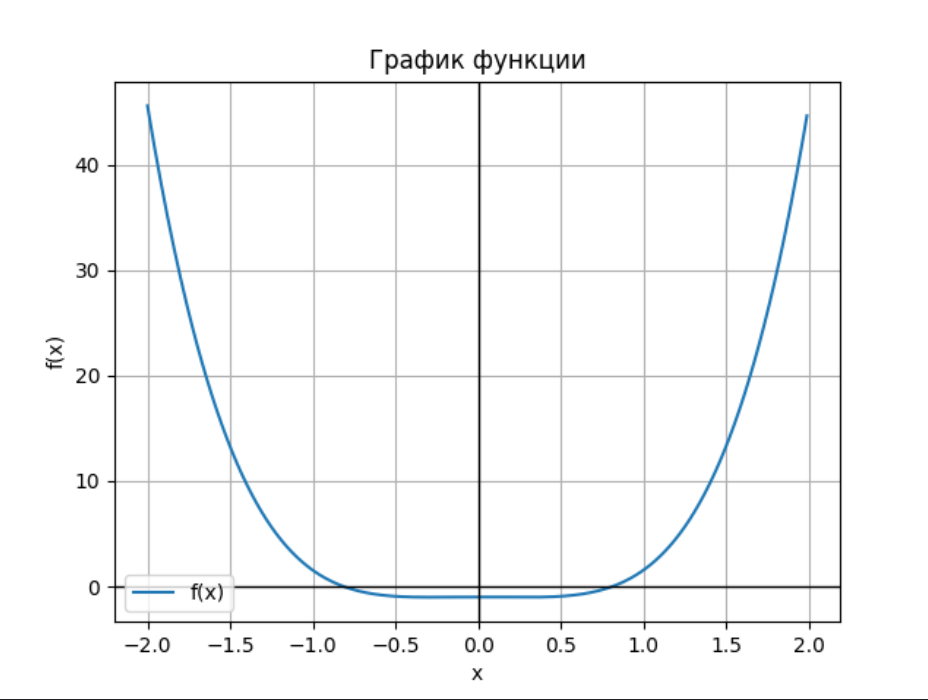
Количество итераций: 3



**Пример №2**. Решение уравнения cos(x) + 3 \* x^4 - 2 методом Ньютона с точностью 0,01, начальное приближение 0:

**Вывод:**

Не удалось найти корень за 1000 итераций.



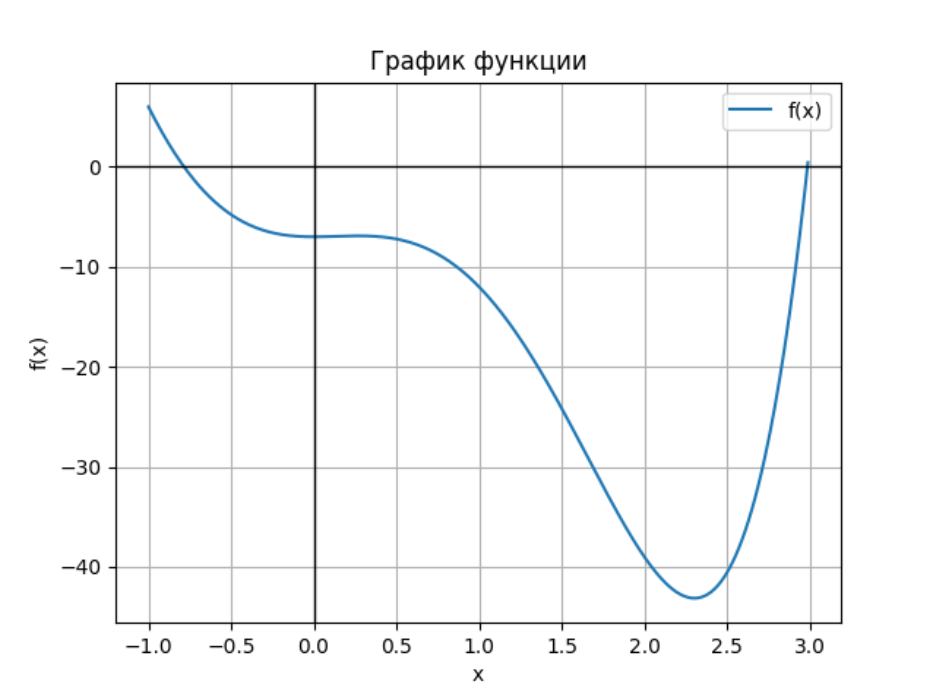
**Пример №3**. Решение уравнения x^5 - 10 \* x^3 + 4 \* x^2 - 7 методом простых итераций с точностью 0,1, начальное приближение 1:

**Вывод:**

Корень: -3.3274839281065893

Значение функции: -2.211954634076349

Количество итераций: 15



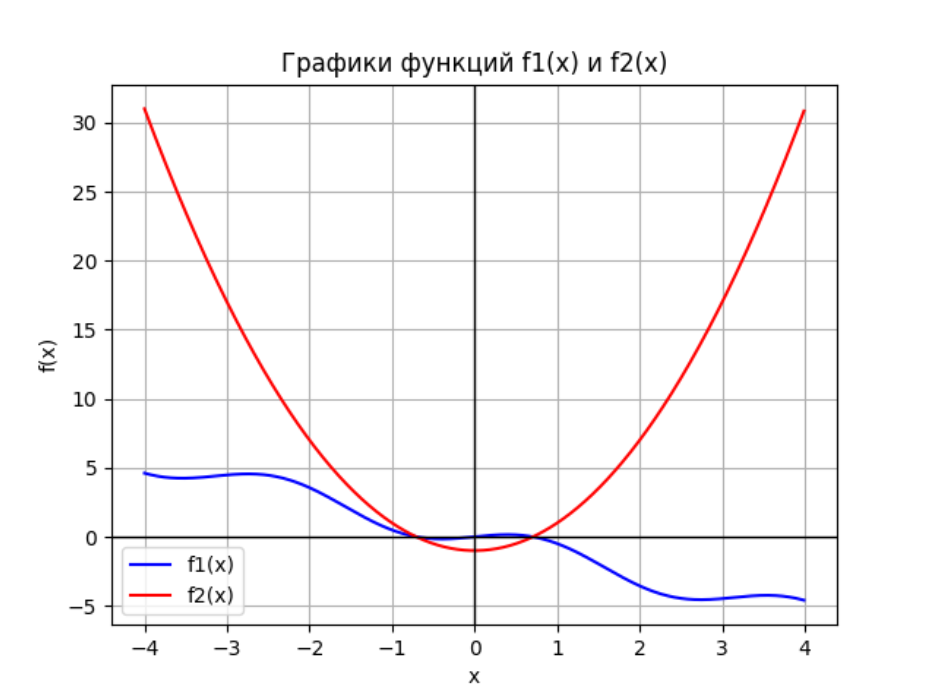
**Пример №4**. Решение системы уравнений f1(x, y) = sin(x + y) - 1.4x, f2(x, y) = x^2 + y ^2 - 1 методом ньютона с точностью 0,01, начальное приближение x=0, y=1:

**Вывод:**

Решение системы: [0.70555676 0.70866245]

Количество итераций: 3

Погрешности: [0.9787986992925334, 0.3503284549711896, 0.08470979029734955, 0.0035781018495097726]



Вывод

В процессе выполнения лабораторной работы я вычислил корни нелинейных уравнений и систем нелинейных уравнений, а также программно реализовал несколько методов для нахождения корней нелинейных уравнений и систем нелинейных уравнений. В частности, были рассмотрены следующие методы: метод хорд, метод Ньютона, метод простых итераций, метод половинного деления и метод секущих.

Я создал программу на языке Python, которая решает задачи нахождения корней нелинейных уравнений и систем с использованием популярных методов. В процессе решения задачи была добавлена возможность визуализации результатов с помощью matplotlib, а также функциональность для записи результатов в файл или вывода их в консоль.