Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

**факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА №2**

по дисциплине

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №7

*Выполнил:*

Студент группы P32091

Кравец Роман Денисович

*Преподаватель:*

Рыбаков Степан Дмитриевич

Изображение выглядит как текст, коллекция картинок

Автоматически созданное описание

Санкт-Петербург, 2023

**Цель работы:**

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

**Порядок выполнения работы:**

Лабораторная работа состоит из двух частей: вычислительной и программной.

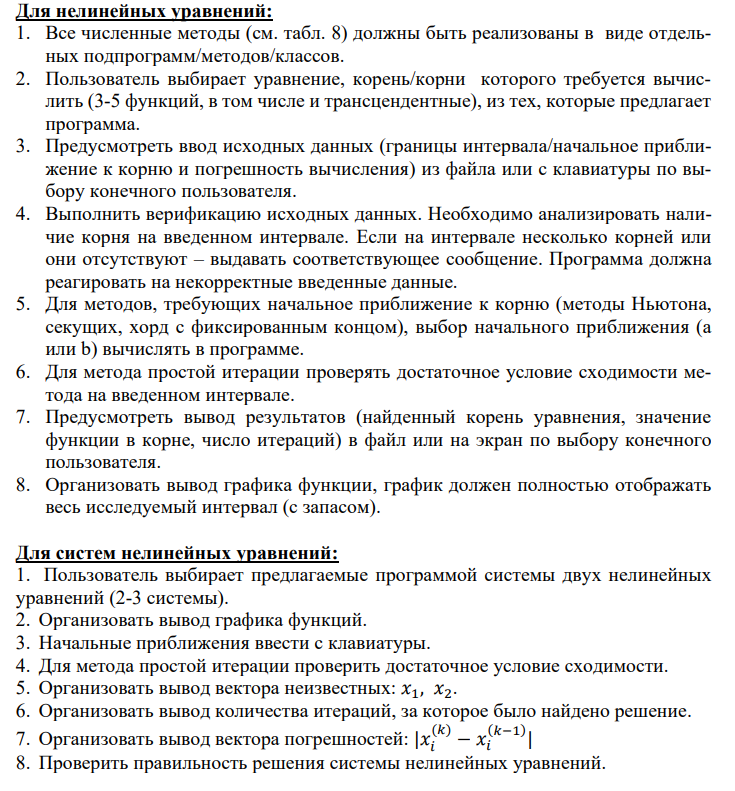
В рамках вычислительной реализации задачи с помощью преобразований данного уравнения:



Были найдены все 3 корня заданного уравнения с помощью трёх методов, среди которых:

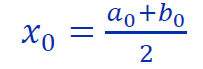
1. Метод половинного деления (для крайнего правого корня).
2. Метод простой итерации (для крайнего левого корня).
3. Метод Ньютона (для центрального корня).

В рамках программной реализации задачи нужно реализовать следующие методы: метод половинного деления, метод секущих, метод простой итерации для решения нелинейных уравнений, а также реализовать метод Ньютона для решения систем нелинейных уравнений.

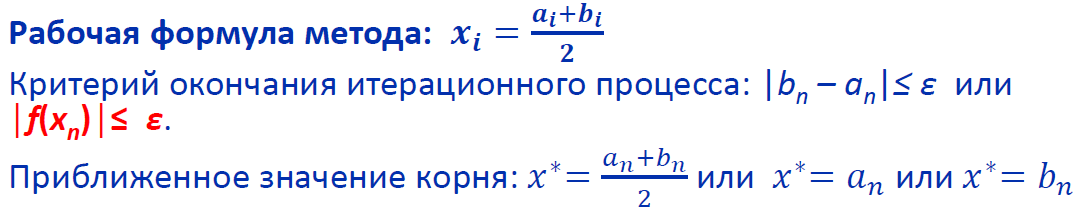


**Описание методов:**

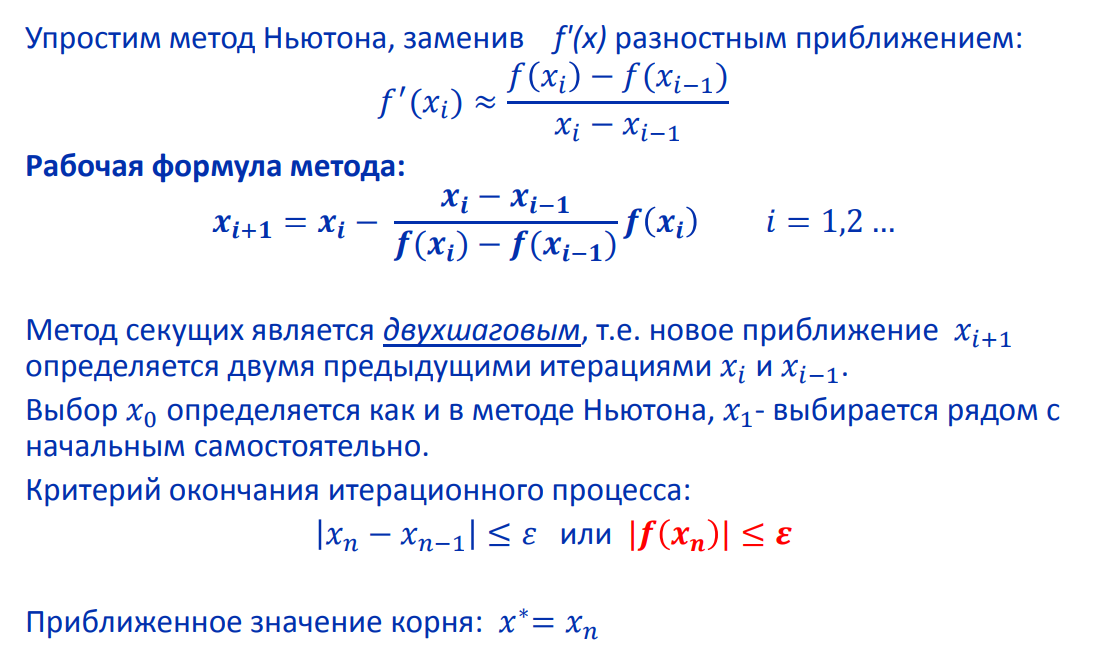
Метод половинного деления (для решения нелинейных уравнений):

Идея метода: начальный интервал изоляции корня делим пополам, получаем начальное приближение к корню:

Вычисляем f(𝑥0). В качестве нового интервала выбираем ту половину отрезка, на концах которого функция имеет разные знаки: [a0, x0] либо [b0, x0]. Другую половину отрезка [a0, b0], на которой функция f(х) знак не меняет, отбрасываем. Новый интервал вновь делим пополам, получаем очередное приближение к корню: х1=(a1+b1)/2.



Метод секущих (для решения нелинейных уравнений):

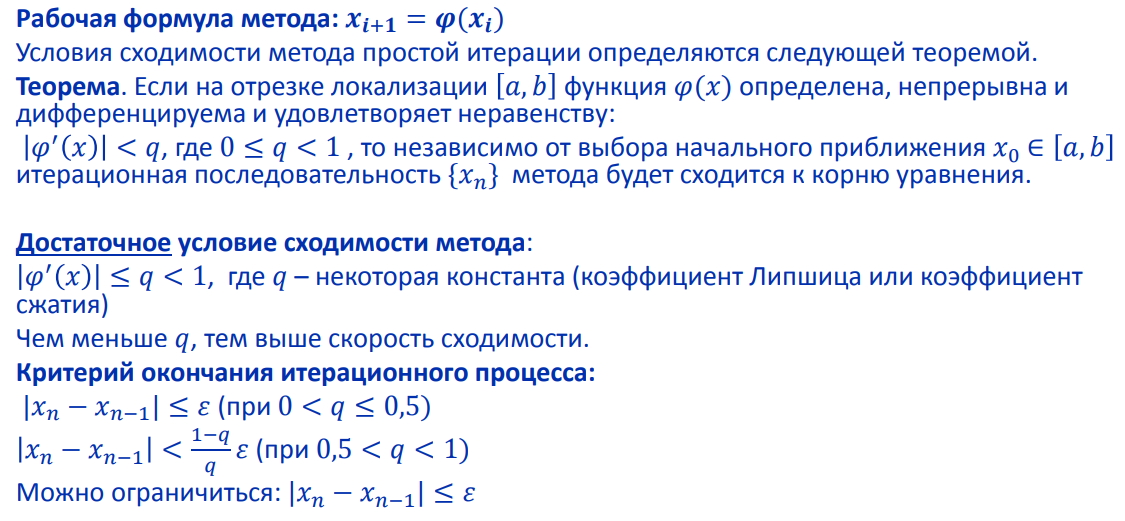


Метод простых итераций (для решения нелинейных уравнений):

Уравнение 𝑓(𝑥)=0 приведем к эквивалентному виду: 𝑥=𝜑(𝑥), выразив 𝑥 из исходного уравнения.

Зная начальное приближение: 𝑥0 ∈ 𝑎, 𝑏, найдем очередные приближения:

𝑥1 = 𝜑(𝑥0) → 𝑥2 = 𝜑(𝑥1) …



Метод Ньютона (для решения систем нелинейных уравнений):

Идея метода:

Изображение выглядит как текст

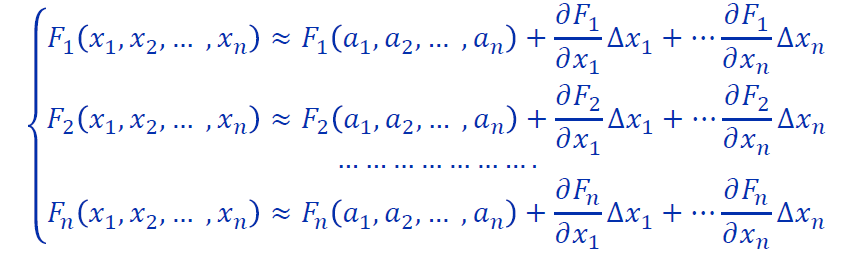
Автоматически созданное описание

К основе метода лежит использование разложения функций 𝐹𝑖 (𝑥1, 𝑥2, …, 𝑥𝑛) в ряд Тейлора в окрестности некоторой фиксированной точки, причем члены, содержащие вторые (и более высоких порядков) производные, отбрасываются.

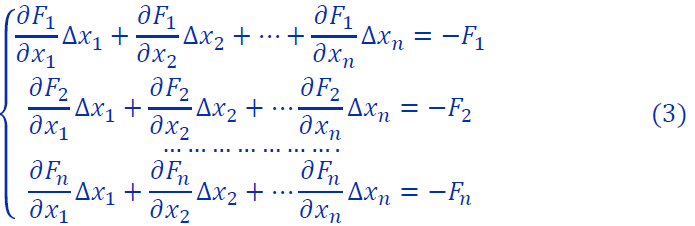
Пусть начальные приближения неизвестных системы (1) получены и равны соответственно 𝑎1, 𝑎­2, …, 𝑎𝑛. Задача состоит в нахождении приращений (поправок) к этим значениям Δ𝑥1, Δ𝑥2, …, Δ𝑥𝑛, благодаря которым решение системы запишется в виде

𝑥1=𝑎1+Δ𝑥1, 𝑥2=𝑎2+Δ𝑥2, …, 𝑥𝑛=𝑎𝑛+Δ𝑥𝑛 (2)

Проведем разложение левых частей уравнений (1) с учетом (2) в ряд Тейлора, ограничиваясь лишь линейными членами относительно приращений:

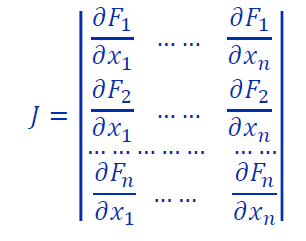


Поскольку в соответствии с (1) левые части этих выражений должны обращаться в нуль, то приравняем к нулю и правые части. Получим следующую систему линейных алгебраических уравнений относительно приращений:



Значения 𝐹1, 𝐹2, …, 𝐹𝑛 и их производные вычисляются при 𝑥1=𝑎1, 𝑥2=𝑎2, …, 𝑥𝑛=𝑎𝑛.

Определителем системы (3) является якобиан:



Итерационный процесс решения систем нелинейных уравнений методом Ньютона состоит в определении приращений Δ𝑥1, Δ𝑥2, …, Δ𝑥𝑛 к значениям неизвестных на каждой итерации.



**Выполнение вычислительной части:**

График функции:

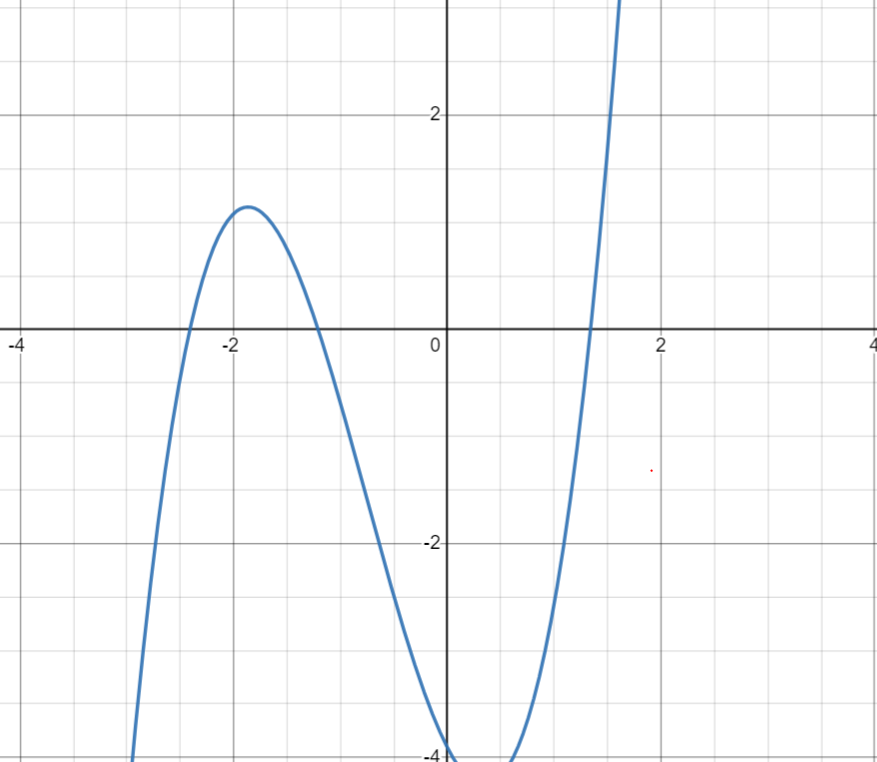


Таблица 1. Уточнение крайнего правого корня методом половинного деления.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № шага | a | b | x | F(a) | F(b) | F(x) | |a-b| |
| 1 | 1.000 | 2.000 | 1.500 | -2.561 | 2.345 | 1.697 | 1.000 |
| 2 | 1.000 | 1.500 | 1.250 | -2.561 | 1.697 | -0.809 | 0.500 |
| 3 | 1.250 | 1.500 | 1.375 | -0.809 | 1.697 | 0.344 | 0.250 |
| 4 | 1.250 | 1.375 | 1.313 | -0.809 | 0.344 | -0.257 | 0.125 |
| 5 | 1.313 | 1.375 | 1.344 | -0.257 | 0.344 | 0.037 | 0.063 |
| 6 | 1.313 | 1.344 | 1.328 | -0.257 | 0.0375 | -0.111 | 0.031 |
| 7 | 1.328 | 1.344 | 1.336 | -0.111 | 0.037 | -0.037 | 0.016 |
| 8 | 1.336 | 1.344 | 1.340 | -0.037 | 0.037 | 0.000 | 0.008 |

Таблица 2. Уточнение крайнего левого корня методом простой итерации.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № итерации | xk | Xk+1 | F(Xk+1) | | Xk+1 - Xk| |
| 1 | -3.000 | -2.597 | -1.022 | 0.403 |
| 2 | -2.597 | -2.507 | -0.485 | 0.09 |
| 3 | -2.507 | -2.464 | -0.259 | 0.043 |
| 4 | -2.464 | -2.441 | -0.145 | 0.023 |
| 5 | -2.441 | -2.428 | -0.084 | 0.013 |
| 6 | -2.428 | -2.421 | -0.051 | 0.007 |

Таблица 3. Уточнение центрального корня методом Ньютона.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № итерации | xk | F(xk) | F`(xk) | Xk+1 | | Xk+1 - Xk| |
| 1 | -1.000 | -0.693 | -3.494 | -1.198 | 0.198 |
| 2 | -1.198 | -0.037 | -3.091 | -1.210 | 0.012 |
| 3 | -1.210 | 0.000 | -3.060 | -1.210 | 0.000 |

**Выполнение программной части:**

**Листинг программы:**

def halfDivision\_method(a, b, function, error):

# Метод половинного деления

counter = 0

middle = (a + b) / 2

counter += 1

while (abs(a - b) > error or abs(function.subs(x, middle)) > error):

a = a if function.subs(x, a) \* function.subs(x, middle) < 0 else middle

b = b if function.subs(x, b) \* function.subs(x, middle) < 0 else middle

middle = (a + b) / 2

counter += 1

return middle, function.subs(x, middle), counter

def secant\_method(a, b, function, error):

# Метод секущих

secondDerivative = secondDiff(function, x)

if function.subs(x, a) \* secondDerivative.subs(x, a) > 0:

x0 = a

else:

x0 = b

x1 = x0 + (b-a)/2

x2 = x1 - (((x1 - x0) / (function.subs(x, x1) - function.subs(x, x0))) \* function.subs(x, x1))

itr = 0

while ((abs(x2 - x1) > error or abs(function.subs(x,x2)) > error) and itr < 1000):

x0 = x1

x1 = x2

x2 =  x1 - (((x1 - x0) / (function.subs(x, x1) - function.subs(x, x0))) \* function.subs(x, x1))

itr += 1

return x2, function.subs(x, x2), itr

def iteration\_method(a, b, function, error):

# Метод простой итерации

counter = 0

firstDerivative = firstDiff(function, x)

if (abs(firstDerivative.subs(x, a)) > abs(firstDerivative.subs(x, b))):

x0 = a

else:

x0 = b

k = -1 / firstDerivative.subs(x, x0)

phi = x + k \* function

x1 = phi.subs(x, x0)

x1Derivative = x1.diff(x)

# проверка на достаточное условие сходимости метода

check = abs(x1Derivative.subs(x, a)) < 1 and abs(x1Derivative.subs(x, b)) < 1

while ((abs(x1 - x0) > error or abs(function.subs(x, x1)) > error) and counter < 1000):

x0 = x1

x1 = phi.subs(x, x0)

counter += 1

print(function.subs(x, x1))

return x1, function.subs(x, x1), counter, check

def newtonMethod(f, g, x0, y0, epsilon):

#метод Ньютона для системы нелинейных уравнений

jacobi\_matrix = [[firstDiff(f, x) , firstDiff(f, y)],

[firstDiff(g, x) , firstDiff(g, y)]]

dx = Symbol("dx")

dy = Symbol("dy")

first = jacobi\_matrix[0][0] \* dx + jacobi\_matrix[0][1] \* dy + f

second = jacobi\_matrix[1][0] \* dx + jacobi\_matrix[1][1] \* dy + g

iter = 0

vector\_errors\_x = []

vector\_errors\_y = []

while (True):

newSystem = [first.subs([(x, x0), (y, y0)]),

second.subs([(x, x0), (y, y0)])]

try:

solutions = linsolve(newSystem, (dx, dy)).args[0]

except IndexError:

return None

x1 = x0 + solutions[0]

y1 = y0 + solutions[1]

iter+=1

vector\_errors\_x.append(abs(x1 - x0))

vector\_errors\_y.append(abs(y1 - y0))

if (abs(x1 - x0) <= epsilon and abs(y1 - y0) <= epsilon):

break

x0 = x1

y0 = y1

return x0, y0, iter, vector\_errors\_x, vector\_errors\_y

**Примеры работы программы:**

1. **Для метода половинного деления**

Лабораторная работа #2 (7)

Численное решение нелинейных уравнений и систем

Взять исходные данные из файла (файл) или ввести с клавиатуры (консоль)?

консоль

Что вы хотите решить? (уравнение / систему)

уравнение

Выберите функцию.

1: x^3 + 2.28x^2 - 1.934x - 3.907

2: x^3 - x + 4

3: sin(x) + 0.1

4: e^x \* sin(x)

5: x^4 + 4x^3 - 22x^2 - 100x - 75

Функция: 1

Выберите метод решения.

1: Метод половинного деления

2: Метод секущих

3: Метод простой итерации

1

Введите границы интервала через пробел(Например: 1 2)

0 2

Выберите погрешность вычисления

0.001

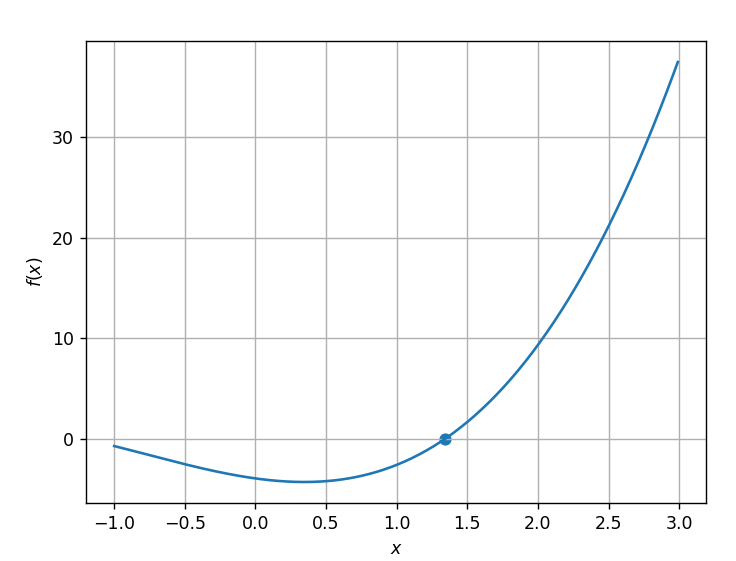
Куда вывести результат работы в файл(файл) или в консоль(консоль)?

консоль

Корень уравнения: 1.33978271484375

Значение функции в корне: -0.000565643975622088

Число итераций: 15



1. **Для метода секущих**

Лабораторная работа #2 (7)

Численное решение нелинейных уравнений и систем

Взять исходные данные из файла (файл) или ввести с клавиатуры (консоль)?

консоль

Что вы хотите решить? (уравнение / систему)

уравнение

Выберите функцию.

1: x^3 + 2.28x^2 - 1.934x - 3.907

2: x^3 - x + 4

3: sin(x) + 0.1

4: e^x \* sin(x)

5: x^4 + 4x^3 - 22x^2 - 100x - 75

Функция: 2

Выберите метод решения.

1: Метод половинного деления

2: Метод секущих

3: Метод простой итерации

2

Введите границы интервала через пробел(Например: 1 2)

-3 -1

Выберите погрешность вычисления

0.001

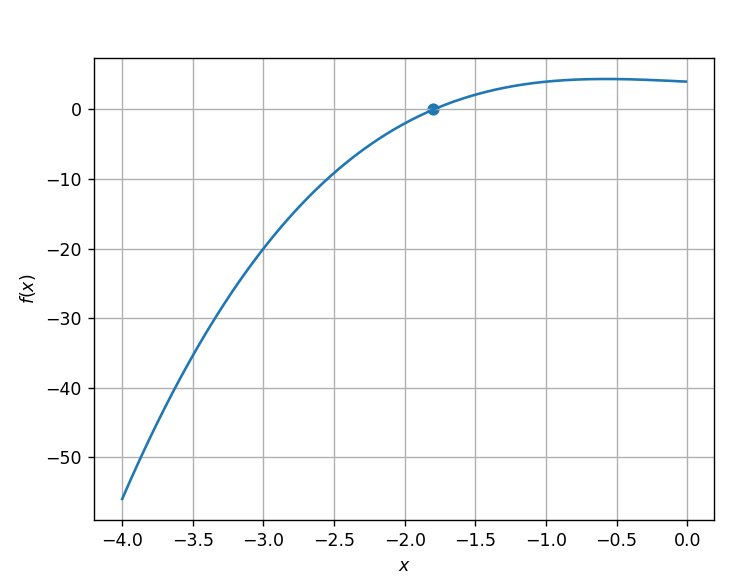
Куда вывести результат работы в файл(файл) или в консоль(консоль)?

консоль

Корень уравнения: -1.79632555056354

Значение функции в корне: -3.16598279788138e-5

Число итераций: 3



1. Для метода простых итераций

Лабораторная работа #2 (7)

Численное решение нелинейных уравнений и систем

Взять исходные данные из файла (файл) или ввести с клавиатуры (консоль)?

консоль

Что вы хотите решить? (уравнение / систему)

уравнение

Выберите функцию.

1: x^3 + 2.28x^2 - 1.934x - 3.907

2: x^3 - x + 4

3: sin(x) + 0.1

4: e^x \* sin(x)

5: x^4 + 4x^3 - 22x^2 - 100x - 75

Функция: 3

Выберите метод решения.

1: Метод половинного деления

2: Метод секущих

3: Метод простой итерации

3

Введите границы интервала через пробел(Например: 1 2)

2 4

Выберите погрешность вычисления

0.01

Куда вывести результат работы в файл(файл) или в консоль(консоль)?

консоль

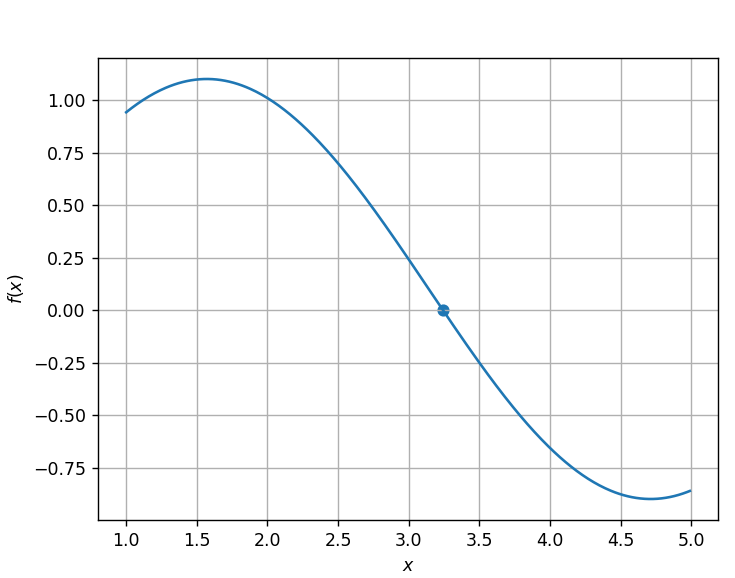
-0.00255345996752478

Условие сходимости выполняется.

Корень уравнения: 3.24432673244233

Значение функции в корне: -0.00255345996752478

Число итераций: 7



1. Для метода Ньютона(для решения систем нелинейных уравнений)

Лабораторная работа #2 (7)

Численное решение нелинейных уравнений и систем

Взять исходные данные из файла (файл) или ввести с клавиатуры (консоль)?

консоль

Что вы хотите решить? (уравнение / систему)

систему

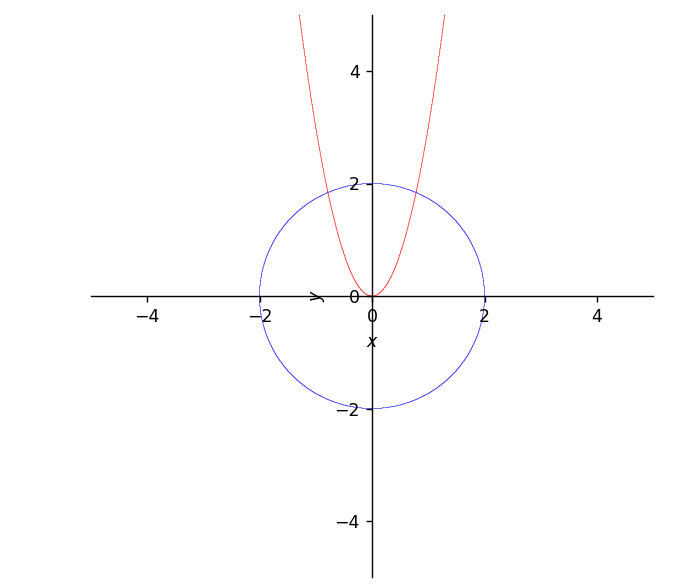
Выберите систему уравнения

1: x^2 + y^2 - 4 = 0; y = 3 \* x^2

2: y + sin(x) - 1 = 0; 2y + sin(x) - 2= 0

3: (x + 2y)(2x - y + 1) - 6 = 0; (2x - y + 1)/(x + 2y) - 2/3 = 0

Система: 1



Выберите начальное приближение(x и y)

Начальное приближение: 2 2

Выберите погрешность вычисления

Погрешность вычисления должна быть положительным числом

0.01

Куда вывести результат работы в файл(файл) или в консоль(консоль)?

консоль

Корень уравнения: 0.785314975286998

Значение функции в корне: 1.84026576315053

Число итераций: 4

Вектор погрешностей х:

0.846153846153846

0.311105667156623

0.0574255114025333

0.00209959663942272

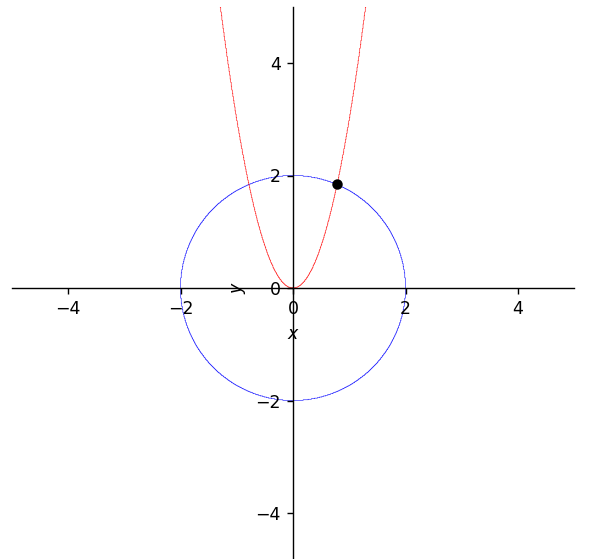
Вектор погрешностей y:

0.153846153846154

0.00587947084762352

8.61215569503138e-6

1.84781079326513e-11



**Вывод**

Во время выполнения лабораторной работы я изучил работу метода половинного деления, метода секущих, метода простых итераций для решения нелинейных уравнений, а также работу метода Ньютона для решения систем нелинейных уравнений, реализовал все эти методы с помощью языка программирования Python.