Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №2**

по «Вычислительной математике»

Решение нелинейных уравнений и систем нелинейных уравнений

Выполнил:

Студент группы P32151

Кортыш А.О.

Преподаватель:

Малышева Т.А.

Санкт-Петербург

2023

**Цель работы.**

Изучить численные методы решения нелинейных уравнений и их си-

стем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов.

**Вычислительная реализация**

### Рабочие формулы используемых методов

Метод Ньютона:

Метод половинного деления:

Метод простой итерации:

Функция:

График 1.

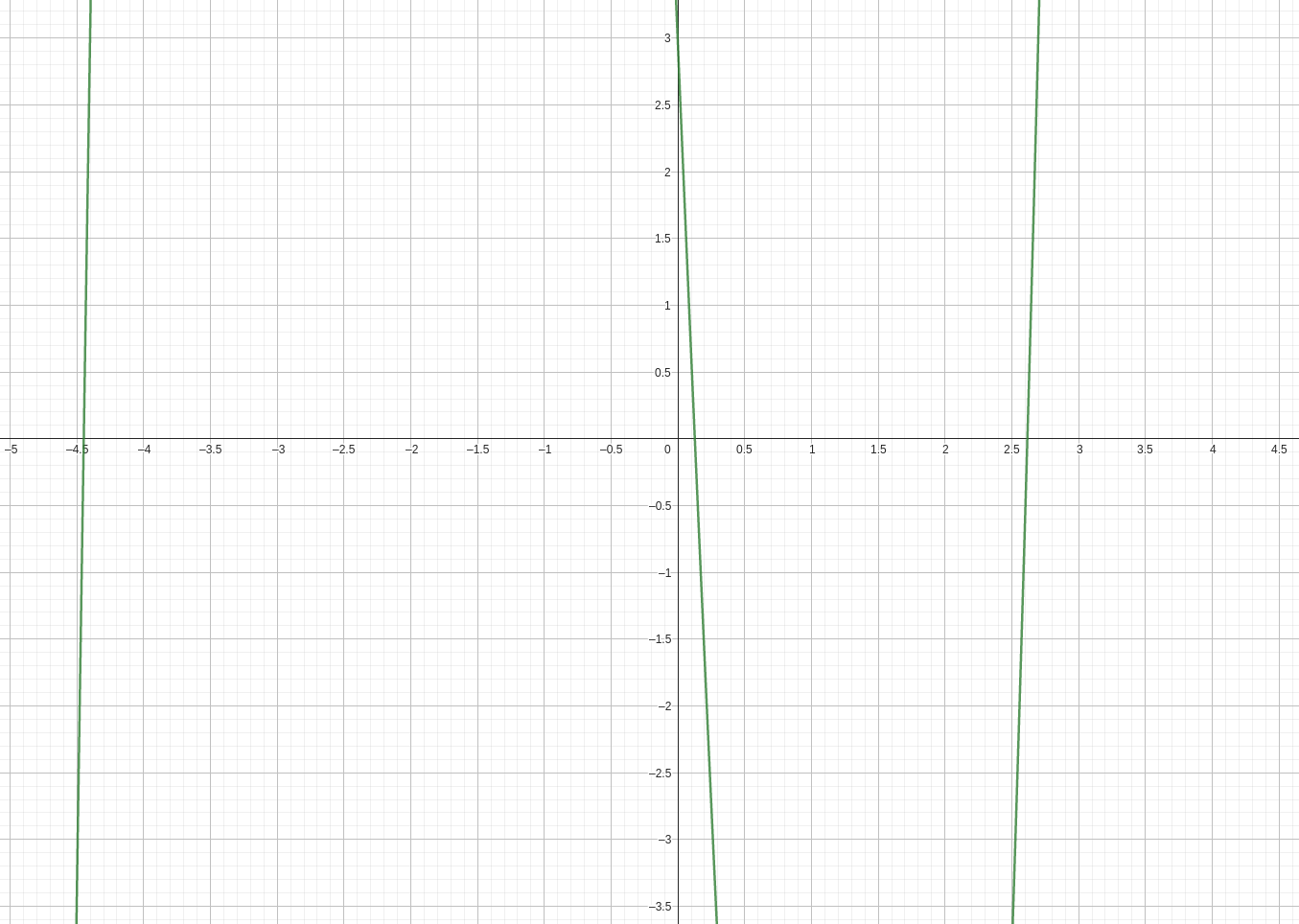


Таблица 1. Метод половинного деления

| № шага | a | b | x | f(a) | f(b) | f(x) | |a-b| |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | -5,000 | -4,000 | -4,500 | -43,100 | 24,470 | -3,417 | 1,000 |
| 2 | -4,500 | -4,000 | -4,250 | -3,417 | 24,470 | 11,907 | 0,500 |
| 3 | -4,500 | -4,250 | -4,375 | -3,417 | 11,907 | 4,602 | 0,250 |
| 4 | -4,500 | -4,375 | -4,438 | -3,417 | 4,602 | 0,683 | 0,125 |
| 5 | -4,500 | -4,438 | -4,469 | -3,417 | 0,683 | -1,345 | 0,063 |
| 6 | -4,469 | -4,438 | -4,453 | -1,345 | 0,683 | -0,325 | 0,031 |
| 7 | -4,453 | -4,438 | -4,445 | -0,325 | 0,683 | 0,180 | 0,016 |

Таблица 2. Метод Ньютона

| № итерации |  | f() | f'() |  | || |
| --- | --- | --- | --- | --- | --- |
| 1 | 3,000 | 16,420 | 50,720 | 2,676 | 0,324 |
| 2 | 2,676 | 2,176 | 37,486 | 2,618 | 0,058 |
| 3 | 2,618 | 0,065 | 35,246 | 2,616 | 0,002 |

Таблица 3. Метод простой итерации

| № итерации |  |  | f() | |-| |
| --- | --- | --- | --- | --- |
| 1 | 0 | 0,124 | 0,126 | 0,1245 |
| 2 | 0,124 | 0,126 | 0,126 | 0,0022 |

**Листинг программы.**

def simple\_iteration(a: int, b: int, epsilon: float, function: Add) -> typing.Tuple[float, float, int]:

iteration\_count: int = 0

callable\_function = lambdify(x, function, 'numpy')

if not check\_common(a, b, callable\_function):

raise ValueError

first\_derivative = lambdify(x, function.diff(x), 'numpy')

coefficient = -1/max(abs(first\_derivative(a)), abs(first\_derivative(b)))

if abs(1+coefficient\*first\_derivative(a)) >= 1 or abs(1+coefficient\*first\_derivative(b)) >= 1:

raise SimpleIterationException

iteration\_function: Callable[[float], float] = lambda y: y + coefficient\*callable\_function(y)

prev\_x = 1e9

cur\_x = a if abs(first\_derivative(a)) > abs(first\_derivative(b)) else b

while abs(callable\_function(cur\_x)) > epsilon and abs(cur\_x - prev\_x) > epsilon:

cur\_x, prev\_x = iteration\_function(cur\_x), prev\_x

iteration\_count += 1

return cur\_x, callable\_function(cur\_x), iteration\_count

def chord\_method(a: int, b: int, epsilon: float, function: Add) -> typing.Tuple[float, float, int]:

iteration\_count: int = 0

callable\_function = lambdify(x, function, 'numpy')

if not check\_common(a, b, callable\_function):

raise ValueError

iteration\_function: Callable[[float, float], float] = lambda c, d: (c \* callable\_function(d) - d \* callable\_function(c))\

/ (callable\_function(d) - callable\_function(c))

cur\_x: float = iteration\_function(a, b)

prev\_x: float = 1e9

while abs(prev\_x - cur\_x) > epsilon and abs(b - a) > epsilon and abs(callable\_function(cur\_x)) > epsilon:

if callable\_function(cur\_x) \* callable\_function(a) < 0:

b = cur\_x

else:

a = cur\_x

prev\_x = cur\_x

cur\_x = iteration\_function(a, b)

iteration\_count += 1

return cur\_x, callable\_function(cur\_x), iteration\_count

def secant\_method(a: int, b: int, epsilon: float, function: Add) -> typing.Tuple[float, float, int]:

iteration\_count: int = 0

callable\_function = lambdify(x, function, 'numpy')

if not check\_common(a, b, callable\_function):

raise ValueError

second\_derivative = lambdify(x, function.diff(x).diff(x), 'numpy')

if callable\_function(a) \* second\_derivative(a) > 0:

prev\_x = a

cur\_x = a+2\*epsilon

else:

prev\_x = b

cur\_x = b - 2\*epsilon

iteration\_function: Callable[[float, float], float] = lambda prev\_y, prev\_prev\_y: prev\_y - callable\_function(prev\_y) \* (prev\_y - prev\_prev\_y) /\

(callable\_function(prev\_y) - callable\_function(prev\_prev\_y))

while abs(cur\_x - prev\_x) > epsilon and abs(callable\_function(cur\_x)) > epsilon:

cur\_x, prev\_x = iteration\_function(cur\_x, prev\_x), cur\_x

iteration\_count += 1

return cur\_x, callable\_function(cur\_x), iteration\_count

def newton\_method(x\_0: float, y\_0: float, epsilon: float, system: typing.Tuple[Add, Add]) -> \

typing.Tuple[typing.Tuple[float, float], int, typing.Tuple[float, float], typing.Tuple[float, float]]:

iteration\_count = 0

cur\_x: float = x\_0

cur\_y: float = y\_0

prev\_x: float = 1e9

prev\_y: float = 1e9

callable\_f: Callable[[float, float], float] = lambdify([x, y], system[0], 'numpy')

callable\_g: Callable[[float, float], float] = lambdify([x, y], system[1], 'numpy')

callable\_f\_diff\_x: Callable[[float, float], float] = lambdify([x, y], system[0].diff(x), 'numpy')

callable\_f\_diff\_y: Callable[[float, float], float] = lambdify([x, y], system[0].diff(y), 'numpy')

callable\_g\_diff\_x: Callable[[float, float], float] = lambdify([x, y], system[1].diff(x), 'numpy')

callable\_g\_diff\_y: Callable[[float, float], float] = lambdify([x, y], system[1].diff(y), 'numpy')

iteration\_function\_y: Callable[[float, float], float] = lambda x1, y1: (callable\_f(x1, y1)\*callable\_g\_diff\_x(x1, y1)/callable\_f\_diff\_x(x1, y1) - callable\_g(x1, y1)) / \

(-callable\_g\_diff\_x(x1, y1)\*callable\_f\_diff\_y(x1, y1)/callable\_f\_diff\_x(x1, y1) + callable\_g\_diff\_y(x1, y1))

iteration\_function\_x: Callable[[float, float, float], float] = lambda x1, y1, delta\_y1: (-callable\_f\_diff\_y(x1, y1) \* delta\_y1 - callable\_f(x1, y1)) / callable\_f\_diff\_x(x1, y1)

while abs(cur\_x - prev\_x) > epsilon or abs(cur\_y - prev\_y) > epsilon:

delta\_y: float = iteration\_function\_y(cur\_x, cur\_y)

delta\_x: float = iteration\_function\_x(cur\_x, cur\_y, delta\_y)

cur\_x, prev\_x, cur\_y, prev\_y = cur\_x + delta\_x, cur\_x, cur\_y + delta\_y, cur\_y

iteration\_count += 1

return tuple([cur\_x, cur\_y]), iteration\_count, tuple([cur\_x - prev\_x, cur\_y - prev\_y]), tuple([callable\_f(cur\_x, cur\_y), callable\_g(cur\_x, cur\_y)])

**Примеры и результат работы программы**

**Пример 1.**

Input:

1 to solve nonlinear equation

2 to solve system of nonlinear equations

1

1: sin(0.5\*x) + cos(0.5\*x) - 0.5

2: 3\*x - exp(x) + 5

3: x\*\*3 - 0.1\*x\*\*2 + 0.5

4: 2\*x - cos(3\*x) - 5

Input function number: 1

Input:

1 to get data interval and error from file

2 to proceed via console

2

Input:

1 to output data to file

2 to output to console

2

Input interval start and end with space between them, for example, 2.1 4:

-10 -8

Input allowed error: 0.0001

Available method names are:

1. Chord

2. Secant

3. Iteration

Input method name: Chord

Method finished in 3 iterations.

Result value is: -8.5767.

Function value of result: 5e-06

### Пример 2

Input:

1 to solve nonlinear equation

2 to solve system of nonlinear equations

2

System #1

-1.2\*x + sin(2\*x - y) - 0.4

x\*\*2 + y\*\*2 - 1

System #2

-0.5\*x\*\*2 + x + y - 0.7

2\*x - 0.166666666666667\*y\*\*3 + y - 1.6

Input system number: 1

Input initial values for x and y with space between them, for example, 2.1 4

100 100

Input allowed error: 0.0001

Results are: x = 0.4728, y = -0.8812

Number of iterations: 295

Errors vector: error\_x = -0.0, error\_y = -0.0

Equations value with result x, y: f = -0.0, g = 0.0

### Пример 3

Input:

1 to solve nonlinear equation

2 to solve system of nonlinear equations

1

1: sin(0.5\*x) + cos(0.5\*x) - 0.5

2: 3\*x - exp(x) + 5

3: x\*\*3 - 0.1\*x\*\*2 + 0.5

4: 2\*x - cos(3\*x) - 5

Input function number: 3

Input:

1 to get data interval and error from file

2 to proceed via console

1

Input:

1 to output data to file

2 to output to console

2

Input filename: test

Incorrect interval

**Выводы.**

Программирование методов для решения нелинейных уравнений и систем нелинейных уравнений не отличается сложностью для большинства методов. С другой стороны проверка корректности использования того или иного метода может быть затратным как для программы, так и для программиста. Особой сложность отличается программирование метода простой итерации, так как нахождение функции, которая удовлетворяет условию сходимости метода не всегда просто.