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: **2**

Преподаватель:
Малышева Татьяна Алексеевна

Выполнил:
Барсуков Максим Андреевич
Группа: P3215

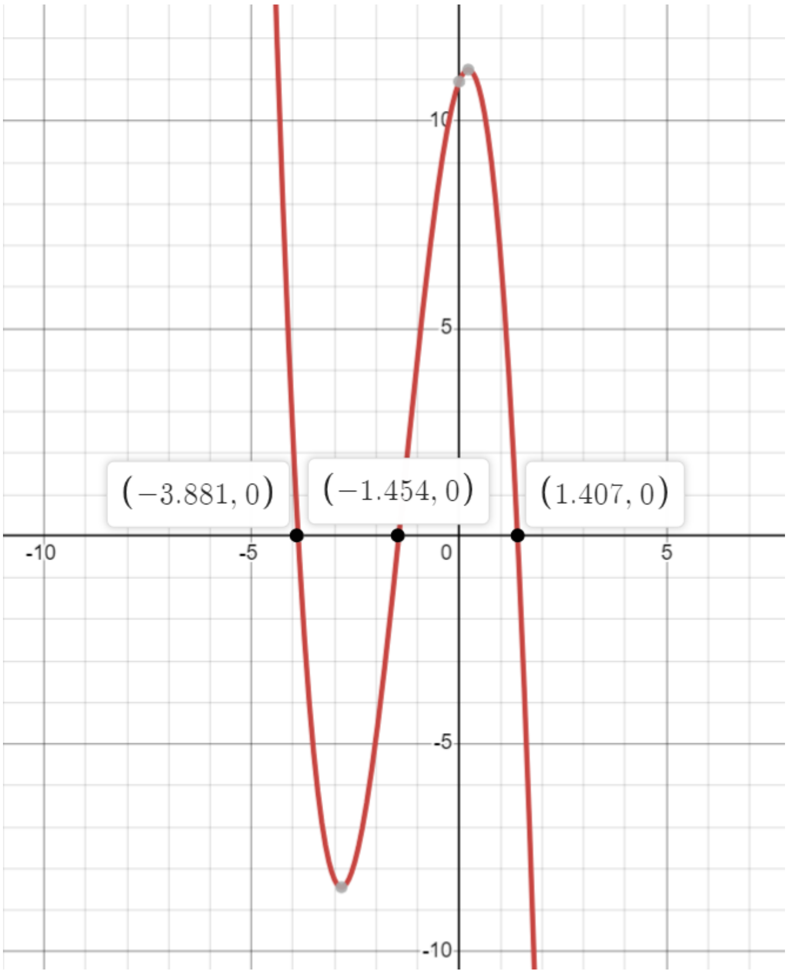
Санкт-Петербург, 2024 г.

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

1. Вычислительная реализация задачи

1. Решение нелинейного уравнения

1. $-1{,}38x^3 - 5{,}42x^2 + 2{,}57x + 10{,}95$



2.

Для определения интервалов изоляции корней данного уравнения, можно воспользоваться методом интервалов знакопеременности. Для этого нужно найти значения функции на различных интервалах и определить знак функции на каждом из них.

Получим приближенные значения корней:
x ≈ -3.9, x ≈ -1.5, x ≈ 1.4

Теперь нужно разбить ось x на 4 интервала: (-∞, -3.9), (-3.9, -1.5), (-1.5, 1.4) и (1.4, +∞). На каждом из этих интервалов нужно определить знак функции.

Для этого можем вычислить значения функции в произвольной точке каждого интервала. Например, для интервала (-∞, -3.9) можно выбрать x = -4, для интервала (-3.9, -1.5) x = -2, для интервала (-1.5, 1.4) x = 0, и для интервала (1.4, +∞) x = 2.

Таким образом, получим следующие значения функции:

для x = -4: f(-4) = 2.27

для x = -2: f(-2) = -4.83

для x = 0: f(0) = 10.95

для x = 2: f(2) = -16.63

Знаки функции на каждом интервале будут соответственно:

(-∞, -3.9)	(-3.9, -1.5)	(-1.5, 1.4)	(1.4, +∞)
+	-	+	-

Таким образом, мы получаем два интервала изоляции корней уравнения:

(-4, -1.5), (-1.5, 1) и (1, 1.5).

3.

$x_1 \approx -3{,}88$
$x_2 \approx -1{,}45$
$x_3 \approx 1{,}41$

4.

Крайний правый корень – Метод простой итерации

№	x_k	x_{k+1}	$f(x_{k+1})$	$\lvert x_{k+1} - x_k \rvert$
1	1.000	1.541	6.720	0.541
2	1.541	1.298	-3.022	0.244
3	1.298	1.470	2.137	0.172
4	1.470	1.360	-1.371	0.111
5	1.360	1.437	0.956	0.077
6	1.437	1.385	-0.637	0.051
7	1.385	1.421	0.439	0.035
8	1.421	1.397	-0.297	0.024
9	1.397	1.413	0.203	0.016
10	1.413	1.402	-0.138	0.011
11	1.402	**1.410**	0.094	0.008

Крайний левый корень – Метод хорд

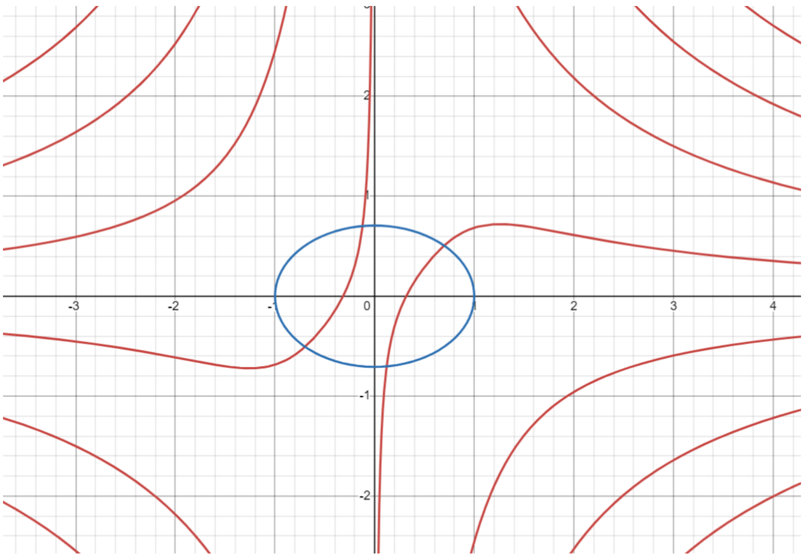
№	a	b	x	f(a)	f(b)	f(x)	$\lvert x_{k+1} - x_k \rvert$
1	-4.000	-1.908	-3.254	2.270	-4.098	-7.253	1.346
2	-4.000	-3.254	-3.822	2.270	-7.253	-0.996	0.568
3	-4.000	-3.822	-3.876	2.270	-0.996	-0.072	0.054
4	-4.000	-3.876	**-3.880**	2.270	-0.072	-0.005	0.004

Центральный корень – Метод половинного деления

№	a	b	x	f(a)	f(b)	f(x)	|a – b|
1	-1.500	1.000	-0.250	-0.443	6.720	9.990	2.500
2	-1.500	-0.250	-0.875	-0.443	9.990	5.476	1.250
3	-1.500	-0.875	-1.188	-0.443	5.476	2.566	0.625
4	-1.500	-1.188	-1.344	-0.443	2.566	1.058	0.312
5	-1.500	-1.344	-1.422	-0.443	1.058	0.305	0.156
6	-1.500	-1.422	-1.461	-0.443	0.305	-0.070	0.078
7	-1.461	-1.422	-1.441	-0.070	0.305	0.117	0.039
8	-1.461	-1.441	-1.451	-0.070	0.117	0.024	0.020
9	-1.461	-1.451	**-1.456**	-0.070	0.024	-0.023	0.010

2. Решение системы нелинейных уравнений

1. $\begin{cases} tg(xy+0.1) = x^2 \\ x^2 + 2y^2 = 1 \end{cases}$, Метод Ньютона



2.

$$\begin{cases} tg(xy+0.1) = x^2 \\ x^2 + 2y^2 = 1 \end{cases} \to \begin{cases} f(x,y) = 0 \\ g(x,y) = 0 \end{cases} \to \begin{cases} tg(xy+0.1) - x^2 = 0 \\ x^2 + 2y^2 - 1 = 0 \end{cases}$$

Отметим, что решение системы уравнений являются точки пересечения эллипса и $tg(xy+0.1)-x^2=0$, следовательно, система имеет не более четырех различных решений.

Построим матрицу Якоби:

$\frac{\partial f}{\partial x}=y\,sec(xy+0.1)-2,\ \frac{\partial f}{\partial y}=x\,sec^2(xy+0.1),\ \frac{\partial g}{\partial x}=2x,\ \frac{\partial g}{\partial y}=4y$

$$\begin{vmatrix} \frac{\partial f(x,y)}{\partial x} & \frac{\partial f(x,y)}{\partial y} \\ \frac{\partial g(x,y)}{\partial x} & \frac{\partial g(x,y)}{\partial y} \end{vmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = -\begin{pmatrix} f(x,y) \\ g(x,y) \end{pmatrix}$$

$$\begin{vmatrix} y\,sec(xy+0.1)-2 & x\,sec^2(xy+0.1) \\ 2x & 4y \end{vmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} x^2-tg(xy+0.1) \\ 1-\ x^2-2y^2 \end{pmatrix}$$

$$\begin{cases} ysec(xy+0.1)\Delta x-2\Delta x+\ x\,sec^2(xy+0.1)\Delta y = x^2-tg(xy+0.1) \\ 2x\Delta x+4y\Delta y=1-\ x^2-2y^2 \end{cases}$$

Корень 1: Шаг 1: Выбираем $x_0=-0.12;\ y_0=0.7$

$$\begin{cases} ysec(xy+0.1)\Delta x-2\Delta x+\ x\,sec^2(xy+0.1)\Delta y = x^2-tg(xy+0.1) \\ 2x\Delta x+4y\Delta y=1-\ x^2-2y^2 \end{cases}$$

Шаг 2. Решаем полученную систему.

$$\begin{cases} \Delta x+\ 0.077\,\Delta y\ =\ 0.0154 \\ -0.2\Delta x+2.8\Delta y=0.01 \end{cases} \rightarrow \Delta x=-0.0014;\ \Delta y=0.0019$$

Шаг 3. Вычисляем очередные приближения:

$x_1=x_0+\Delta x=-0.12-0.0014=-0.1214$

$y_1=y_0+\Delta y=0.7+0.0019=0.7019$

$|x_1-x_0|\le \varepsilon,\ |y_1-y_0|\le \varepsilon$

$|-0.1214+0.12|\le \varepsilon,\ |0.7019-0.7|\le \varepsilon \rightarrow$ ответ найден, **корень 1**: $(-0.1214, 0.7019)$

Аналогично находим **другой корень**: $(0.698, 0.506)$

Из графического решения, корни симметричны, следовательно, **другие 2 корня** $(-0.698, -0.506), (0.1214, -0.7019)$

2. Программная реализация задачи

Метод хорд:

```
class ChordMethod(Method):
    name = 'Метод хорд'

    def check(self):
        root_exists = self.equation.root_exists(self.left, self.right)
        return root_exists, 'Отсутствует корень на заданном промежутке' if not 
root_exists else ''
```

```
    def solve(self) -> Result:
        f = self.equation.function
        a = self.left
        b = self.right
        epsilon = self.epsilon
        iteration = 0

        x = a - (b - a) * f(a) / (f(b) - f(a))

        iteration = 0
        last_x = x

        while True:
            if np.abs(f(x)) < epsilon:
                break

            iteration += 1

            if f(a) * f(x) < 0:
                b = x
            else:
                a = x

            x = a - (b - a) * f(a) / (f(b) - f(a))
            if self.log:
                print(f'{iteration}: a = {a:.3f}, b = {b:.3f}, x = {x:.3f}, '
                      f'f(a) = {f(a):.3f}, f(b) = {f(b):.3f}, f(x)={f(x):.3f},
|x_k+1 - x_k| = {abs(x - last_x):.3f}')
            last_x = x

        return Result(x, f(x), iteration, self.decimal_places)
```

Метод Ньютона:

```
dx = 0.00001

class NewtonMethod(Method):
    name = 'Метод Ньютона'

    def solve(self) -> Result:
        f = self.equation.function
        x0 = self.left

        epsilon = self.epsilon
        iteration = 0

        while True:
            iteration += 1
```

```python
            df = derivative(f, x0, dx=dx)
            x1 = x0 - f(x0) / df
            if self.log:
                print(f'{iteration}: x_k = {x0:.3f}, f(x_k) = {f(x0):.3f}, '
                f'f\'(x_k) = {df:.3f}, x_k+1 = {x1:.3f}, |x_k+1 - x_k| = {abs(x1
- x0)}')

            if abs(x1 - x0) < epsilon:
                break
            x0 = x1

        return Result(x1, f(x1), iteration, self.decimal_places)
```

Метод простой итерации:

```python
dx = 0.00001
steps = 100

class SimpleIterationsMethod(Method):
    name = 'Метод простой итерации'

    def __init__(self, equation: Equation, left: float, right: float,
                 epsilon: float, decimal_places: int, log: bool):
        super().__init__(equation, left, right, epsilon, decimal_places, log)
        f = self.equation.function
        max_derivative = max(derivative(f, self.left, dx), derivative(f,
self.right, dx))
        _lambda = - 1 / max_derivative
        self.phi = lambda x: x + _lambda * f(x)

    def check(self):
        if not self.equation.root_exists(self.left, self.right):
            return False, 'Отсутствует корень на заданном промежутке'

        # Достаточное условие сходимости метода |phi'(x)| < 1
        print('phi\'(a) = ', abs(derivative(self.phi, self.left, dx)))
        print('phi\'(b) = ', abs(derivative(self.phi, self.right, dx)))
        for x in numpy.linspace(self.left, self.right, steps, endpoint=True):
            if abs(derivative(self.phi, x, dx)) >= 1:
                return False, 'Не выполнено условие сходимости метода |phi\'(x)|
< 1 на интервале'
        return True, ''

    def solve(self) -> Result:
        f = self.equation.function

        prev = self.left

        iteration = 0
```

```
        while True:
            iteration += 1
            x = self.phi(prev)

            diff = abs(x - prev)
            if self.log:
                print(f'{iteration}: xk = {prev:.3f}, f(xk) = {f(prev):.3f}, '
                      f'xk+1 = φ(xk) = {x:.3f}, |xk - xk+1| = {diff:.3f}')

            if diff <= self.epsilon:
                break
            prev = x
        return Result(x, f(x), iteration, self.decimal_places)
```

Метод простой итерации для систем нелинейных уравнений:

```
def solve(system, x0, y0, epsilon, max_iter=1_000):
    def jacobian(xy):
        x, y = xy
        return np.array([[2*x, 2*y], [2*x, -1]])

    xy = np.array([x0, y0], dtype=float)
    for i in range(max_iter):
        J_inv = np.linalg.inv(jacobian(xy))
        F = np.array(system(xy))
        xy_next = xy - np.dot(J_inv, F)
        error = np.linalg.norm(xy_next - xy)

        if error < epsilon:
            return xy_next, i + 1, error

        xy = xy_next

    raise Exception("Solution not found in {} iterations".format(max_iter))
```

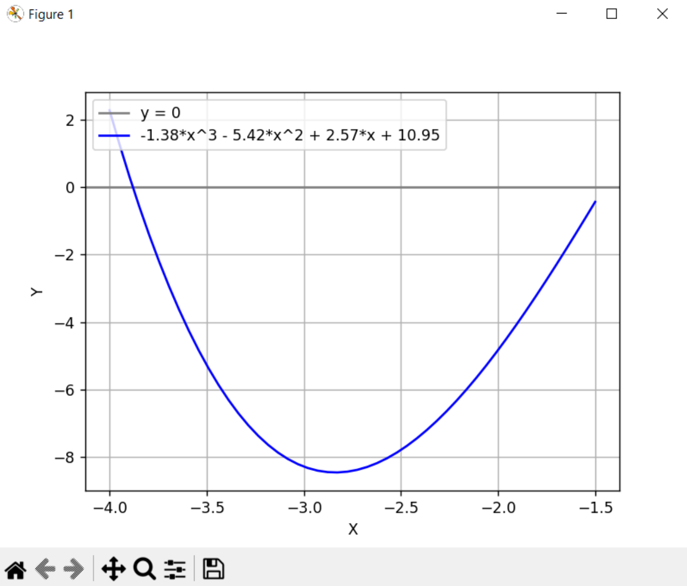
Результаты выполнения программы при различных исходных данных:

```
Выберите тип программы:
1: Нелинейное уравнение
2: Система нелинейных уравнений
Введите номер типа: 1
Выберите уравнение:
1: -1.38*x^3 - 5.42*x^2 + 2.57*x + 10.95
2: x^3 - 1.89*x^2 - 2*x + 1.76
3: x/2 - 2*(x + 2.39)^(1/3)
4: -x/2 + e^x + 5*sin(x)
Введите номер уравнения: 1
Выберите метод:
```

1: Метод половинного деления
2: Метод хорд
3: Метод простой итерации
4: Метод Ньютона
Введите номер метода: 2
Введите имя файла для загрузки исходных данных и интервала или пустую строку, чтобы ввести вручную:
Введите левую границу интервала: -4
Введите правую границу интервала: -1.5
Введите погрешность вычисления: 0.000001
Введите имя файла для вывода результата или пустую строку, чтобы вывести в консоль:
Процесс решения:
1: a = -4.000, b = -1.908, x = -3.254, f(a) = 2.270, f(b) = -4.098, f(x)=-7.253, |x_k+1 - x_k| = 1.346
2: a = -4.000, b = -3.254, x = -3.822, f(a) = 2.270, f(b) = -7.253, f(x)=-0.996, |x_k+1 - x_k| = 0.568
3: a = -4.000, b = -3.822, x = -3.876, f(a) = 2.270, f(b) = -0.996, f(x)=-0.072, |x_k+1 - x_k| = 0.054
4: a = -4.000, b = -3.876, x = -3.880, f(a) = 2.270, f(b) = -0.072, f(x)=-0.005, |x_k+1 - x_k| = 0.004
5: a = -4.000, b = -3.880, x = -3.880, f(a) = 2.270, f(b) = -0.005, f(x)=-0.000, |x_k+1 - x_k| = 0.000
6: a = -4.000, b = -3.880, x = -3.881, f(a) = 2.270, f(b) = -0.000, f(x)=-0.000, |x_k+1 - x_k| = 0.000
7: a = -4.000, b = -3.881, x = -3.881, f(a) = 2.270, f(b) = -0.000, f(x)=-0.000, |x_k+1 - x_k| = 0.000
8: a = -4.000, b = -3.881, x = -3.881, f(a) = 2.270, f(b) = -0.000, f(x)=-0.000, |x_k+1 - x_k| = 0.000

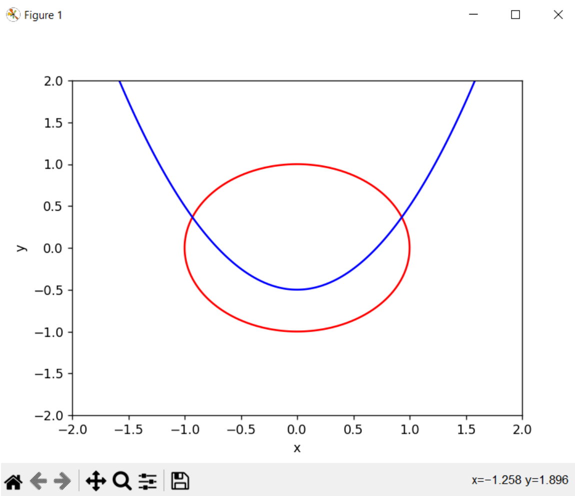
Результат:
Найденный корень уравнения: -3.880518
Значение функции в корне: -1.047914928165028e-07
Число итераций: 8

Еще раз? [y/n]



Выберите тип программы:
1: Нелинейное уравнение
2: Система нелинейных уравнений
Введите номер типа: 2
Выберите систему уравнений:
1: x^2 + y^2 - 1, x^2 - y - 0.5
2: x^2 + y^2 - 1, x - y^2
Введите номер системы: 1
Введите начальные приближения x0, y0: 1 1
Введите погрешность вычисления: 0.000001

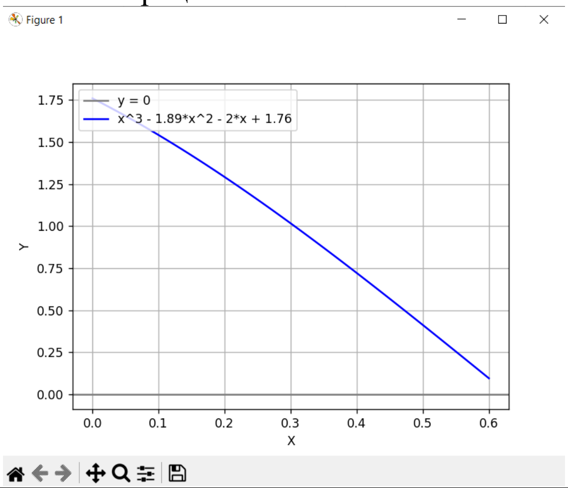
Вектор неизвестных: x1 = 0.93060, x2 = 0.36603
Количество итераций: 5
Вектор погрешностей: 2.33995e-09
Проверка решения системы уравнений:
Невязки: 2.22045e-16, 1.11022e-16



Выберите тип программы:
1: Нелинейное уравнение
2: Система нелинейных уравнений
Введите номер типа: 1
Выберите уравнение:
1: -1.38*x^3 - 5.42*x^2 + 2.57*x + 10.95
2: x^3 - 1.89*x^2 - 2*x + 1.76
3: x/2 - 2*(x + 2.39)^(1/3)
4: -x/2 + e^x + 5*sin(x)
Введите номер уравнения: 2
Выберите метод:
1: Метод половинного деления
2: Метод хорд
3: Метод простой итерации
4: Метод Ньютона
Введите номер метода: 4
Введите имя файла для загрузки исходных данных и интервала или пустую строку, чтобы ввести вручную:
Введите начальное приближение: 0
Введите погрешность вычисления: 0.00001
Введите имя файла для вывода результата или пустую строку, чтобы вывести в консоль:
Процесс решения:

1: x_k = 0.000, f(x_k) = 1.760, f'(x_k) = -2.000, x_k+1 = 0.880, |x_k+1 - x_k| = 0.8800000000430849
2: x_k = 0.880, f(x_k) = -0.782, f'(x_k) = -3.003, x_k+1 = 0.620, |x_k+1 - x_k| = 0.2604368674003591
3: x_k = 0.620, f(x_k) = 0.033, f'(x_k) = -3.190, x_k+1 = 0.630, |x_k+1 - x_k| = 0.010408116407403245
4: x_k = 0.630, f(x_k) = -0.000, f'(x_k) = -3.191, x_k+1 = 0.630, |x_k+1 - x_k| = 7.096700637143627e-07

Результат:
Найденный корень уравнения: 0.62997
Значение функции в корне: 2.220446049250313e-16
Число итераций: 4



Вывод

В ходе выполнения лабораторной работы были изучены численные методы решения нелинейных уравнений и систем нелинейных уравнений с использованием Python. В результате работы были найдены корни заданных уравнений и систем с использованием различных численных методов, а также были построены графики функций для полного представления исследуемых интервалов.