ITMO UNIVERSITY

ИТМО ВТ

# Отчет по Лабораторной работе №2
# по курсу “Вычислительная математика”

## Вариант №5 ( Метод Гаусса )

Выполнил:
Студент группы P32082
Панин Иван Михайлович

Преподаватель:
Машина Екатерина Алексеевна

Санкт-Петербург, 2023

1. Задание лабораторной работы.
2
Лабораторная работа No2
Численное решение нелинейных уравнений и систем
Цель работы: изучить численные методы решения нелинейных уравнений и их си-
стем, найти корни заданного нелинейного уравнения/системы нелиней-ных уравнений, вы-
полнить программную реализацию методов.
No варианта определяется как номер в списке группы согласно ИСУ.
Лабораторная работа состоит из двух частей: вычислительной и про-граммной.
1 Вычислительная реализация задачи:
Задание:
1. Отделить корни заданного нелинейного уравнения графически (вид уравнения
представлен в табл. 6)
2. Определить интервалы изоляции корней.
3. Уточнить корни нелинейного уравнения (см. табл. 6) с точностью ε=10-2
.
4. Используемые методы для уточнения каждого из 3-х корней многочлена пред-
ставлены в таблице 7.
5. Вычисления оформить в виде таблиц (1-5), в зависимости от заданного метода.
Для всех значений в таблице удержать 3 знака после запятой.
5.1 Для метода половинного деления заполнить таблицу 1.
5.2 Для метода хорд заполнить таблицу 2.
5.3 Для метода Ньютона заполнить таблицу 3.
5.4 Для метода секущих заполнить таблицу 4.
5.5 Для метода простой итерации заполнить таблицу 5.
6. Заполненные таблицы отобразить в отчете
2 Программная реализация задачи:
Для нелинейных уравнений:
1. Все численные методы (см. табл. 8) должны быть реализованы в виде отдель-
ных подпрограмм/методов/классов.
2. Пользователь выбирает уравнение, корень/корни которого требуется вычис-
лить (3-5 функций, в том числе и трансцендентные), из тех, которые предлагает
программа.
3. Предусмотреть ввод исходных данных (границы интервала/начальное прибли-
жение к корню и погрешность вычисления) из файла или с клавиатуры по вы-
бору конечного пользователя.
4. Выполнить верификацию исходных данных. Необходимо анализировать нали-
чие корня на введенном интервале. Если на интервале несколько корней или

они отсутствуют – выдавать соответствующее сообщение. Программа должна реагировать на некорректные введенные данные.

5. Для методов, требующих начальное приближение к корню (методы Ньютона, секущих, хорд с фиксированным концом), выбор начального приближения (a или b) вычислять в программе.
6. Для метода простой итерации проверять достаточное условие сходимости метода на введенном интервале.
7. Предусмотреть вывод результатов (найденный корень уравнения, значение функции в корне, число итераций) в файл или на экран по выбору конечного пользователя.
8. Организовать вывод графика функции, график должен полностью отображать весь исследуемый интервал (с запасом).

Для систем нелинейных уравнений:

1. Пользователь выбирает предлагаемые программой системы двух нелинейных уравнений (2-3 системы).
2. Организовать вывод графика функций.
3. Начальные приближения ввести с клавиатуры.
4. Для метода простой итерации проверить достаточное условие сходимости.
5. Организовать вывод вектора неизвестных: $x_1$, $x_2$.
6. Организовать вывод количества итераций, за которое было найдено решение.
7. Организовать вывод вектора погрешностей: $|x_i^{(k)} - x_i^{(k-1)}|$
8. Проверить правильность решения системы нелинейных уравнений.

4

3 Оформить отчет, который должен содержать:

1. Титульный лист.
2. Цель лабораторной работы.
3. Порядок выполнения работы.
4. Рабочие формулы используемых методов.
5. Графики функций на исследуемом интервале.
6. Заполненные таблицы вычислительной части лабораторной работы (в зависимости от варианта: табл. 1 – 5).
7. Листинг программы, по крайней мере, коды используемых методов.
8. Результаты выполнения программы при различных исходных данных.
9. Выводы

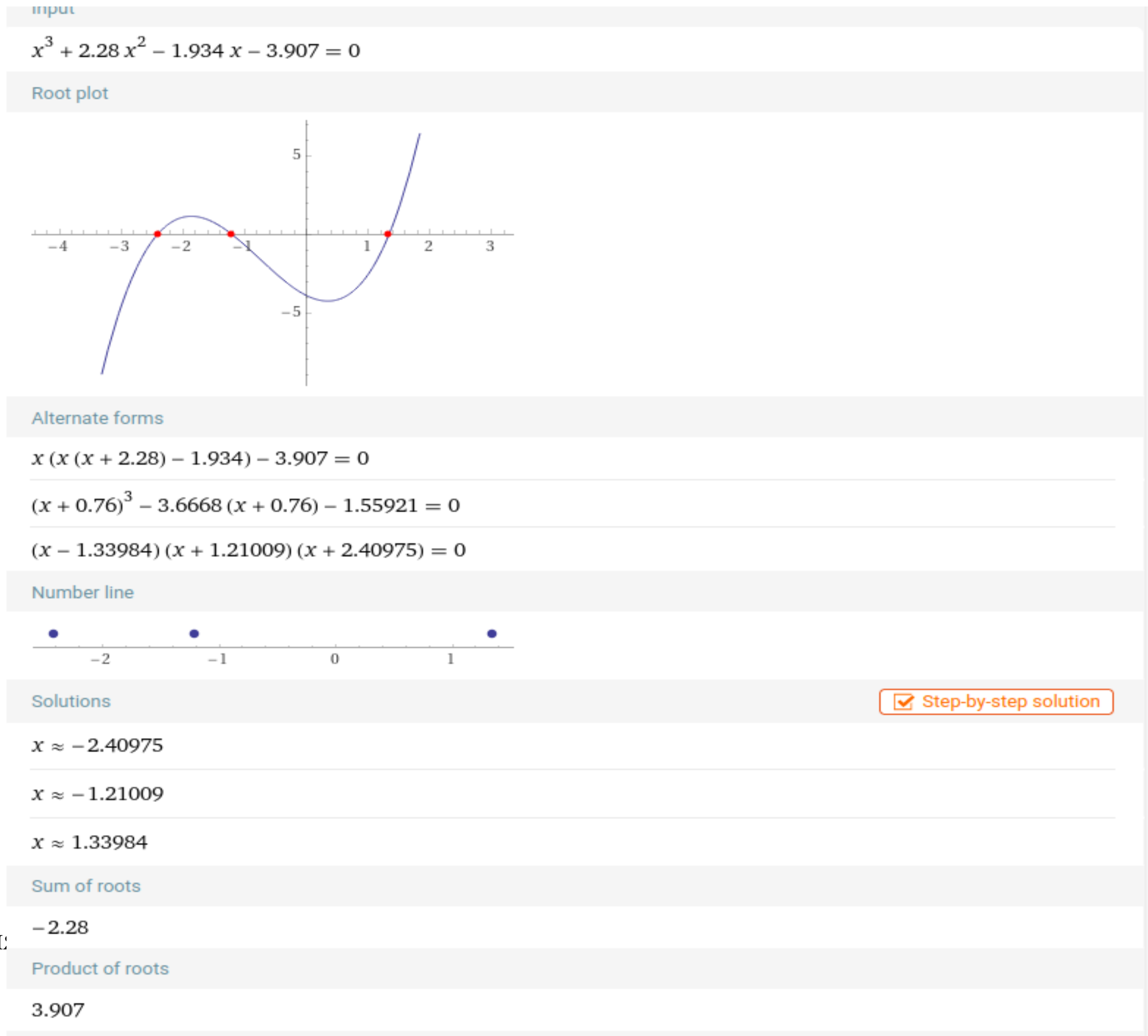
# 2. Цель лабораторной работы.

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

3. Мой вариант
Вариант №7
Уравнение: $x$^3 + 2,28$x$^2 − 1,934$x$ − 3,907

| № Варианта | Крайний правый корень | Крайний левый корень | Центральный корень |
|---|---|---|---|
| 7 | 1 | 5 | 3 |
| 7 | Метод простых итераций | Метод половинного деления | Метод Ньютона |

4. Решение




1) … деле-
ни:

Рабочая формула: [a0 ; b0] — исходный отрезок, x0 принадлежит [a0 ; b0]
x0 = (a0 + b0)/2, а критерий окончания: |bn — an| <= ε или |f(xn)| <= ε
Возьмём ε = 0.01
На графике видно, что этот корень лежит между -3 и -2

| № итерации | a | b | x | F(a) | F(b) | F(x) | \|a-b\| |
|---|---|---|---|---|---|---|---|
| 0 | -3 | -2 | -2.5 | -4,59 | 1,08 | -0,45 | 1 |
| 1 | -2.5 | -2 | -2.25 | -0,45 | 1,08 | 0,6 | 0.5 |
| 2 | -2,5 | -2,25 | -2,375 | -0,45 | 0,6 | 0,15 | 0,25 |
| 3 | -2,5 | -2,375 | -2,4375 | -0,45 | 0,15 | -0,13 | 0.125 |
| 4 | -2,4375 | -2,375 | -2,40625 | -0,13 | 0,15 | 0,02 | 0,0625 |

Таким образом, мы нашли крайний левый корень x* = -2,40625

2) Теперь методом Ньютона найдём центральный корень
Идея метода: y = f(x) на [a:b] заменяется касательной и в качестве x* = xn принимается точка пересечения касательной с осью y.
Рабочая формула: xi = x(i-1) — f(x(i-1))/f'(x(i-1))

Возьмём точность 0.01

f'(x) = 3*x^2 + 4,56 * x — 1,934

| № итерации | xn | f(xn) | f'(xn) | xn+1 | \|xn+1 — xn\| |
|---|---|---|---|---|---|
| 0 | -1 | -0,69 | -3,49 | −1,197707736 | 0,197707736 |
| 1 | −1,197707736 | -0,04 | -3,09 | −1,21065272 | 0,012944984 |

Таким образом, нашли центральный корень x* = -1,2106

3) Далее найдём крайний правый корень методом простых итераций:

Возьмём a = 1 b = 2

Приведём уравнение к виду f1(x) = x

x = x + L*f(x) = f1(x), f1`(x) = 1 + L*f `(x)

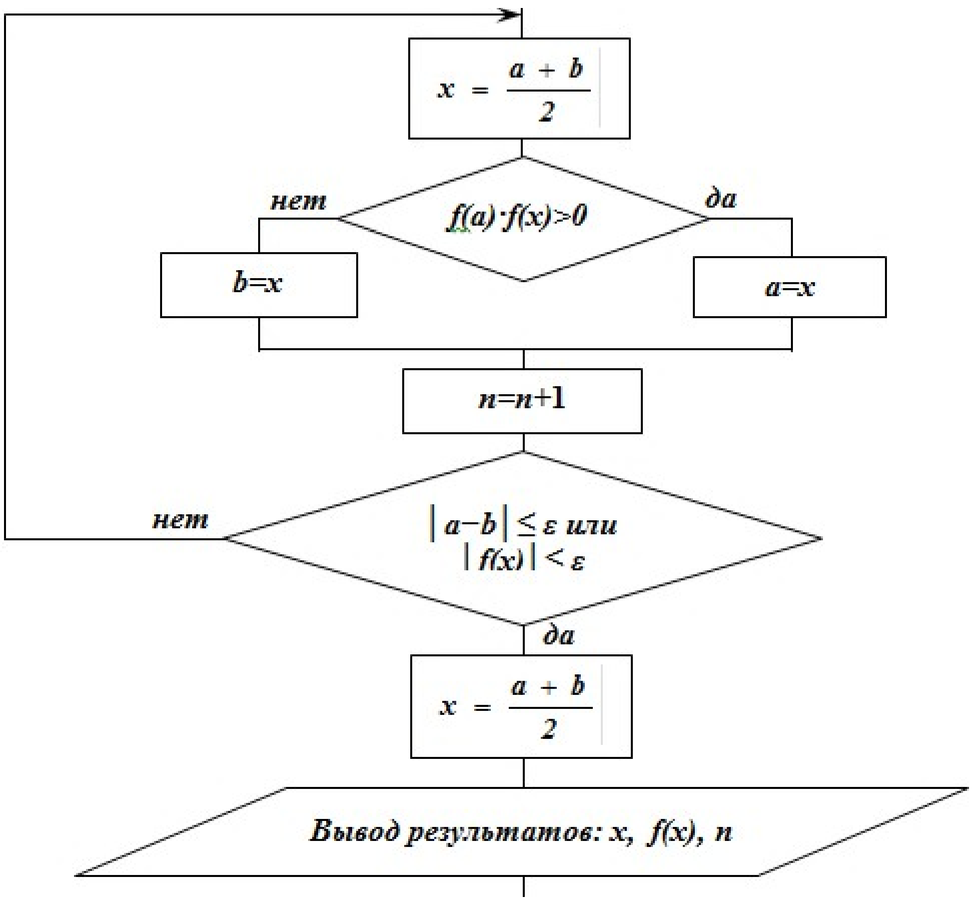
L = -1/(max(a, b) |f `(x)|

f `(2) = 19,186
f `(1) = 5,626
L = 1 / 19,186

x = x + Lx = x^3/19,186 + 2,28x^2/19,186 — 1,934x/19,186 — 3,907/19,186

| № итерации | xi | xi+1 | f1(xi+1) | f(x+1) | \|xi+1-xi\| |
|---|---|---|---|---|---|
| 0 | 2,0 | 1,44 | 0,03 | 0,62 | 0.1 |
| 1 | 1,44 | 1,349 | 0.004 | 0,09 | 0.01 |
| 2 | 1,349 | 1,345 | 0.001 | 0,02 | 0.01 |
| 3 | 1,345 | 1,342 | 0.001 | 0,05 | 0.001 |
| 4 | 1,342 | 1,342 | 0.001 | 0.03 | 0.002 |
| 5 | 1,340 | 1,340 | 0.001 | 0.02 | 0.0001 |
| 6 | 1,339 | 1,339 | 0 | 0 | 0 |

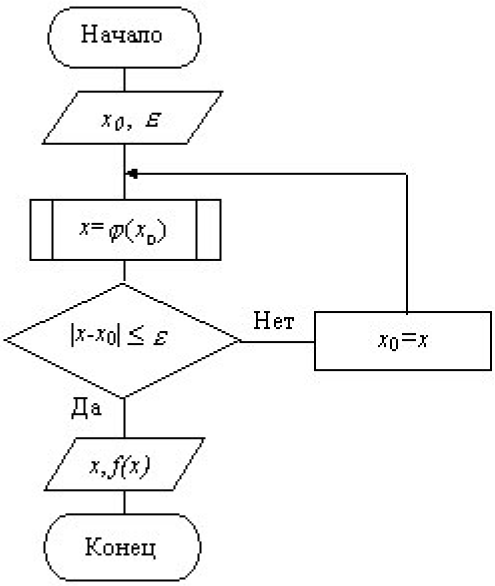
Метод половинного деления.
Начальный интервал изоляции корня делим
пополам, получаем начальное приближение к корню: xo = (ao+bo)/2
И продолжим так делить до тех пор, пока |bn-an|>epsilon, конечно же, не забывая про
теорему у существовании корня. (она поможет понять, какой границе интервала присвоить
значение xi)



Пример работы метода:
Enter command(s1 - system 1, s2 - system 2, f1 - function 1, f2 - function 2, f3 - function 3):
f1
Select method(half division - hd, Newthon - n, simple iteration - si)hd
Your interval(<a> <b>):-3 -1
f(a):%f -20.0
f(b):%f 4.0
Enter epsilon!
0.001
-1.7958984375

```python
def half_division(func, a, b):

        if(check_interval(func, a, b) == False):
                return "Bad interval!"

        xnpy = np.linspace(a, b, 100)
        ynpy = func(xnpy)
        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)
        ax.spines['left'].set_position('center')
        ax.spines['bottom'].set_position('center')
        ax.spines['right'].set_color('none')
        ax.spines['top'].set_color('none')
        ax.xaxis.set_ticks_position('bottom')
        ax.yaxis.set_ticks_position('left')
        plt.plot(xnpy,ynpy, 'g')
        plt.show()

        eps = float(input("Enter epsilon!\n").replace(",", "."))
        x = (a + b)/2

        cur_eps = abs(a-b)
        while(cur_eps > eps):
                x = (a + b)/2
                if func(a) * func(x) > 0 > func(b) * func(x):
                        a = x
                elif func(b) * func(x) > 0 > func(a) * func(x):
                        b = x
                elif func(x) == 0:
                        return x
                else:
                        sys.exit("something wrong in the hd method\n")
                cur_eps = abs(a-b)
        return x
```

Метод простой итерации
Уравнение $f\ x$ = 0 приведем к эквивалентному виду: $x = \varphi(x)$, выразив $x$ из исходного
уравнения.
Зная начальное приближение: $x0 \in a, b$ , найдем очередные приближения:
$x1 = \varphi(x0) \rightarrow x2 = \varphi$
$x1$ ...
Важен выбор начального приближения к корню из малой окрестности для сходимости.
Достаточное условие сходимости метода:
$|\varphi'(x)| \leq q < 1$, где $q$ – некоторая константа (коэффициент Липшица или коэффициент
сжатия)
Чем меньше $q$, тем выше скорость сходимости.



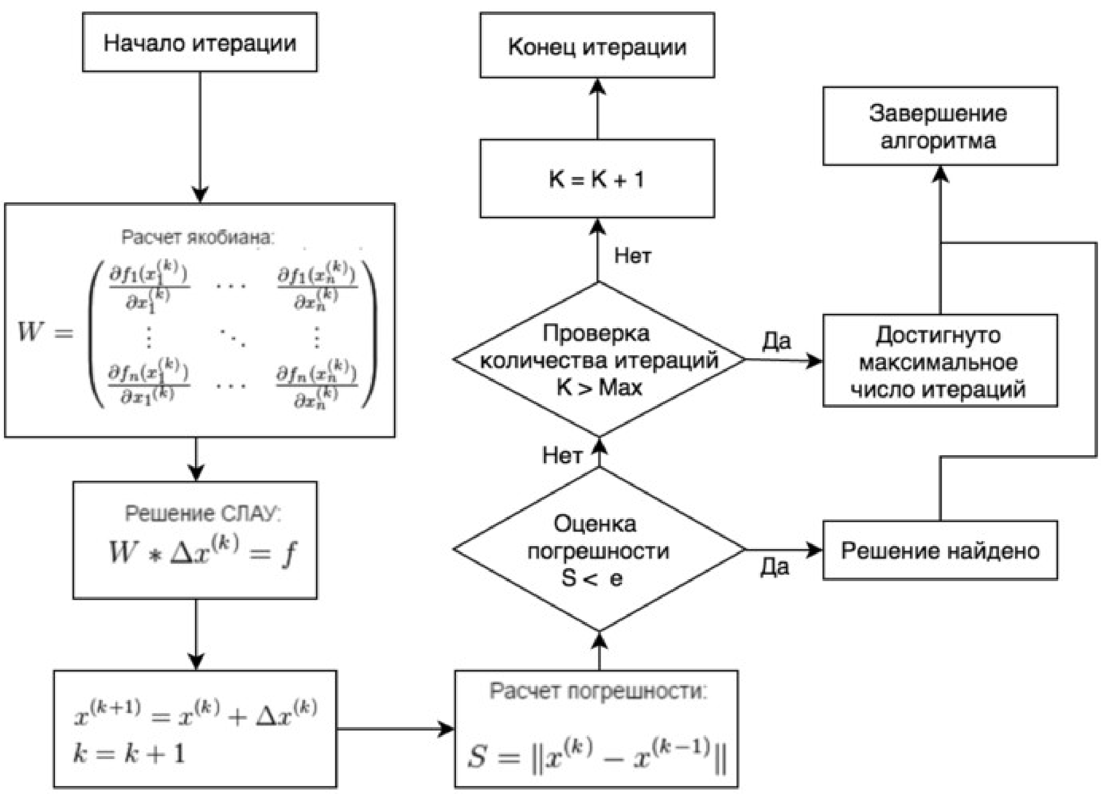

```
def simple_iter(func, a, b):
        # graph somehow

        if(check_interval(func, a, b) == False):
                return "Bad interval!"

        xnpy = np.linspace(a, b, 100)
        ynpy = func(xnpy)
        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)
        ax.spines['left'].set_position('center')
        ax.spines['bottom'].set_position('center')
        ax.spines['right'].set_color('none')
        ax.spines['top'].set_color('none')
```

```
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
plt.plot(xnpy,ynpy, 'g')
plt.show()

eps = float(input("Enter epsilon!\n").replace(",", "."))
deriv_a = (func(a + eps/100)-func(a))/(eps/100)
deriv_b = (func(b + eps/100)-func(b))/(eps/100)
if(deriv_a > deriv_b):
        max_deriv = deriv_a
        xi = a
else:
        max_deriv = deriv_b
        xi = b
lamda = -1 / max_deriv

while(abs(xi + lamda * func(xi) - xi) > eps):
                xi = xi + lamda * func(xi)
return xi + lamda * func(xi)
```

Метод Ньютона.

В 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 $\Delta x1$, $\Delta x2$, … , $\Delta xn$, благодаря которым решение системы запишется в виде $x1=a1+\Delta x1$ , $x2=a2+\Delta x2$ ,…,$xn=an+\Delta xn$



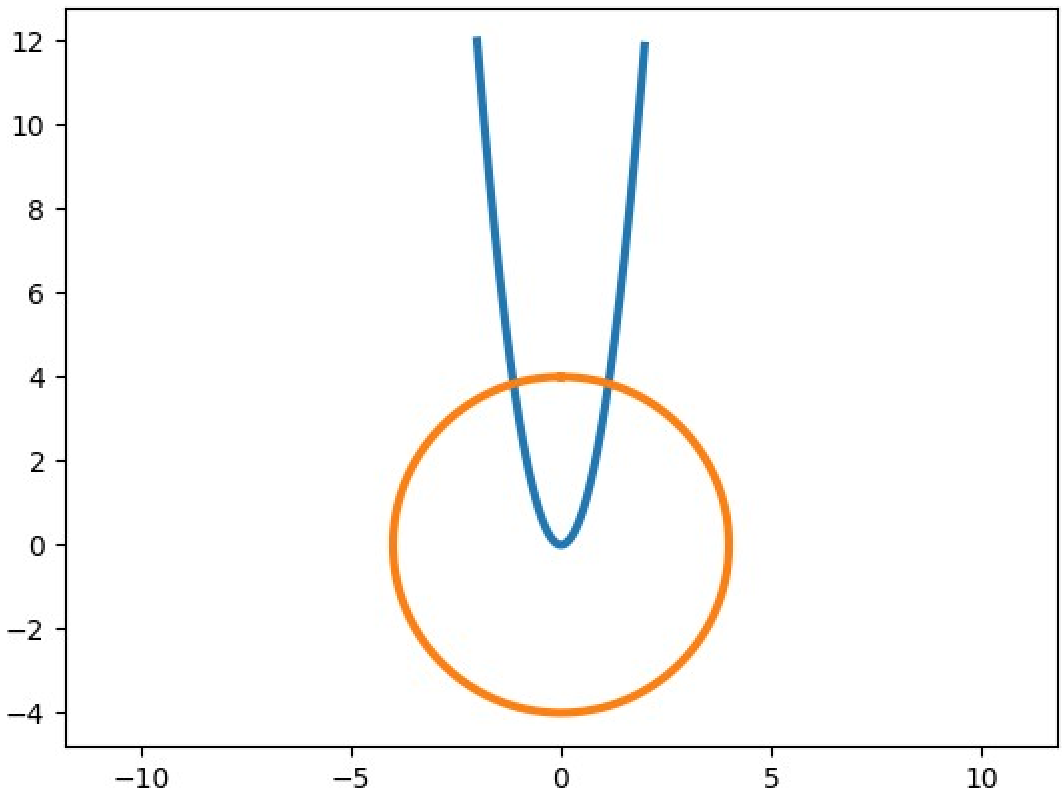
```python
def newton(func, a, b):
        # graph somehow

        if(check_interval(func, a, b) == False):
                return "Bad interval!"

        xnpy = np.linspace(a, b, 100)
        ynpy = func(xnpy)
        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)
        ax.spines['left'].set_position('center')
        ax.spines['bottom'].set_position('center')
        ax.spines['right'].set_color('none')
        ax.spines['top'].set_color('none')
        ax.xaxis.set_ticks_position('bottom')
        ax.yaxis.set_ticks_position('left')
        plt.plot(xnpy,ynpy, 'g')
        plt.show()

        eps = float(input("Enter epsilon!\n").replace(",", "."))

        xip = b
        xi = xip - func(xip)/((func(xip + eps/100)-func(xip))/(eps/100))

        while(abs(func(xi)) <= eps):
                tmp = xi
                xi = xip - func(xip)/((func(xip + eps/100)-func(xip))/(eps/100))
                xip = tmp
        return xi
```

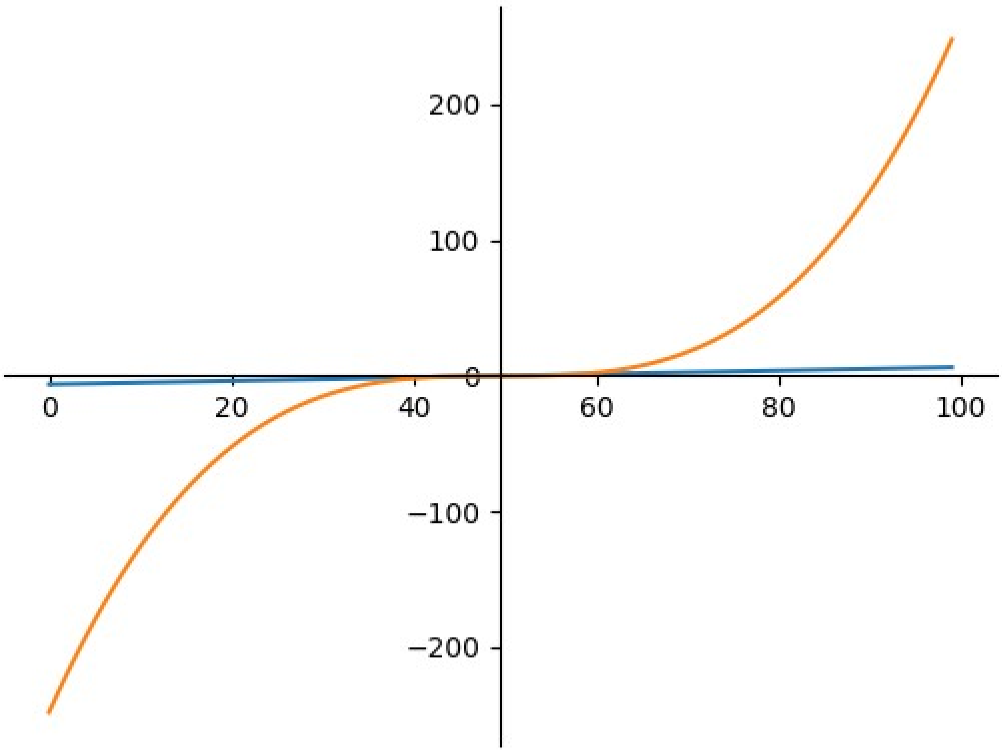
Реализация Систем нелинейных уравнений:

```python
def system1():
        print("x^2 + y^2 - 4 = 0")
```

```python
        print("3x^2 - y = 0")
        eps = float(input("Enter epsilon!\n").replace(",", "."))


        a = np.arange(-2, 2, 0.01)
        t = np.arange(0, 2 * np.pi, 0.01)
        r = 4
        plt.plot(a, 3 * a * a, r * np.sin(t), r * np.cos(t), lw=3)
        plt.axis('equal')
        plt.show()

        xp = float(input("Enter X0!\n").replace(",", "."))
        yp = float(input("Enter Y0!\n").replace(",", "."))

        itersys1(xp, yp, eps)


def get_matr1(x, y):
        return [[2 * x, 2 * y], [-6 * x, 1], [4 - x * x - y * y, 3 * x * x - y]]

def itersys1(x, y, eps):
        dx, dy = np.linalg.solve(get_matr1(x, y)[0:2], get_matr1(x, y)[2])
        xi = x + dx
        yi = y + dy

        while(abs(x - xi) > eps or abs(y - yi) > eps):
                itersys1(xi, yi, eps)
                return

        x = xi
        y = yi
        print("X:", x, "Y:", y)


def system2():
        print("y/(1 + y*y) - 2x = 0")
        print("x/(1+x*x) - 2y = 0")

        eps = float(input("Enter epsilon!\n").replace(",", "."))

        x = np.linspace(-np.pi, np.pi, 100)
        t = np.linspace(-np.pi / 1000, np.pi / 1000, 100)

        y = 2*x + 2 * x * t * t -t
        z = 2 * x + 2 * x * y * y - y

        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)
        ax.spines['left'].set_position('center')
        ax.spines['bottom'].set_position('center')
        ax.spines['right'].set_color('none')
```

```
        ax.spines['top'].set_color('none')
        ax.xaxis.set_ticks_position('bottom')
        ax.yaxis.set_ticks_position('left')


        plt.plot(y)
        plt.plot(z)

        plt.show()

        xp = float(input("Enter X0!\n").replace(",", "."))
        yp = float(input("Enter Y0!\n").replace(",", "."))

        itersys2(xp, yp, eps)

def get_matr2(x, y):
         return [[2 + 2 * y * y, 4 * x * y - 1],[4 * x * y - 1, 2 + 2 * x * x],
[-2 * x - 2 * x * y * y + y, -2 * y - 2 * y * x * x + x]]


def itersys2(x, y, eps):
        dx, dy = np.linalg.solve(get_matr2(x, y)[0:2], get_matr2(x, y)[2])
        xi = x + dx
        yi = y + dy

        while(abs(x - xi) > eps or abs(y - yi) > eps):
                itersys2(xi, yi, eps)
                return

        print("X:", xi, " Y:", yi)
        return
```






Выводы:
В ходе лабораторной работы Я попробовал различные методы решения нелинейных уравнений. Также я научился реализовывать программно и вручную методы для решения нелинейных уравнений.