Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «**Вычислительной математике**»

Системы нелинейных уравнений

Выполнил:

Студент группы P3233

Нгуен Нгок Дык

Преподаватели:

Перл О.В.

Санкт-Петербург

2022

I. Описание метода.

A. Решение нелинейных уравнений

1. Метод деления пополам

Суть метода заключается в том, что на каждой итераций отрезок на котором мы ищем решение уменьшается в 2 раза.

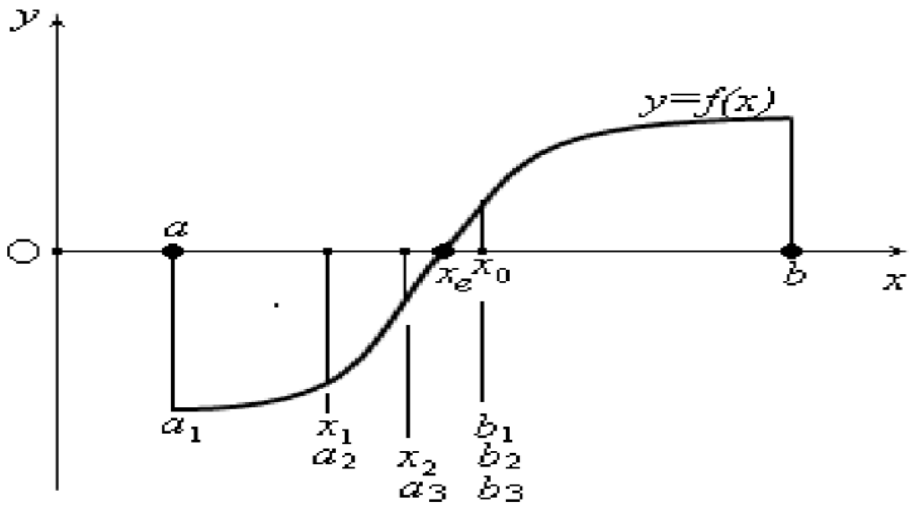
Рабочая формула метода: $x_i = \frac{a_i + b_i}{2}$

Для нахождения корня уравнения. Будем последовательно сужать отрезок поиска, и когда заданная точность будет достигнута результат будет найден.

Метод половинного деления всегда сходится. Скорость сходимости линейна.

Критерий окончания итерационного процесса: $|b_n - a_n| \leq \varepsilon$ или $|f(x_n)| \leq \varepsilon$

Приближенное значение корня: $x^* = \frac{a_n + b_n}{2}$ или $x^* = b_n$



2. Метод простой итераций

Для применения этого метода исходное нелинейное уравнение $f(x) = 0$ заменяют эквивалентным:

$$3.\quad x = \phi(x)$$

Пусть на отрезке $[a, b]$ расположен единственный корень. Примем за x_0 любое значение из интервала $[a, b]$. Вычислим значение функции $\phi(x)$ при $x = x_0$ и найдем уточненное значение $x_1 = \phi(x_0)$. Продолжая этот процесс неограниченно, получим последовательность приближений к корню:

$x_{n+1} = \phi x_n$

Сходимость: Если функция $\phi(x)$ определена и непрерывна на интервале $[a, b]$ и $|\phi'(x)| < 1$, $x \in [a, b]$, то процесс итераций сходится с любой точностью при любом начальном значении x_0 изинтервала $[a, b]$.

В. Решение систем нелинейных уравнений

1. **Метод Ньютона**

Рассмотрим систему нелинейных уравнений

$$F(x) = 0,\ F(x)\ ,\ x \in R^n$$

И предположим, что существует вектор $X \in R^n$, являющийся решением системы

Будем считать, что $F(x) = (f_1(x),\ f_2(x),\ \dots\ ,\ f_n(x))^T$, причём $f_1() \in C^1(D) \forall i$.

Разложим $F(x)$ в окрестности точки X:

$F(x) = F(x^0) + F'(x^0)(x - x^0) + o(||\, x - x^0||)$. Здесь

$$F'(x) = \frac{\partial F(x)}{\partial x} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1}, & \frac{\partial f_1(x)}{\partial x_2}, & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1}, & \frac{\partial f_2(x)}{\partial x_2}, & \cdots & \frac{\partial f_2(x)}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n(x)}{\partial x_1}, & \frac{\partial f_n(x)}{\partial x_2}, & \cdots & \frac{\partial f_n(x)}{\partial x_n} \end{bmatrix}$$

Называется матрицей Якоби, а её определитель - якобианом системы (б).

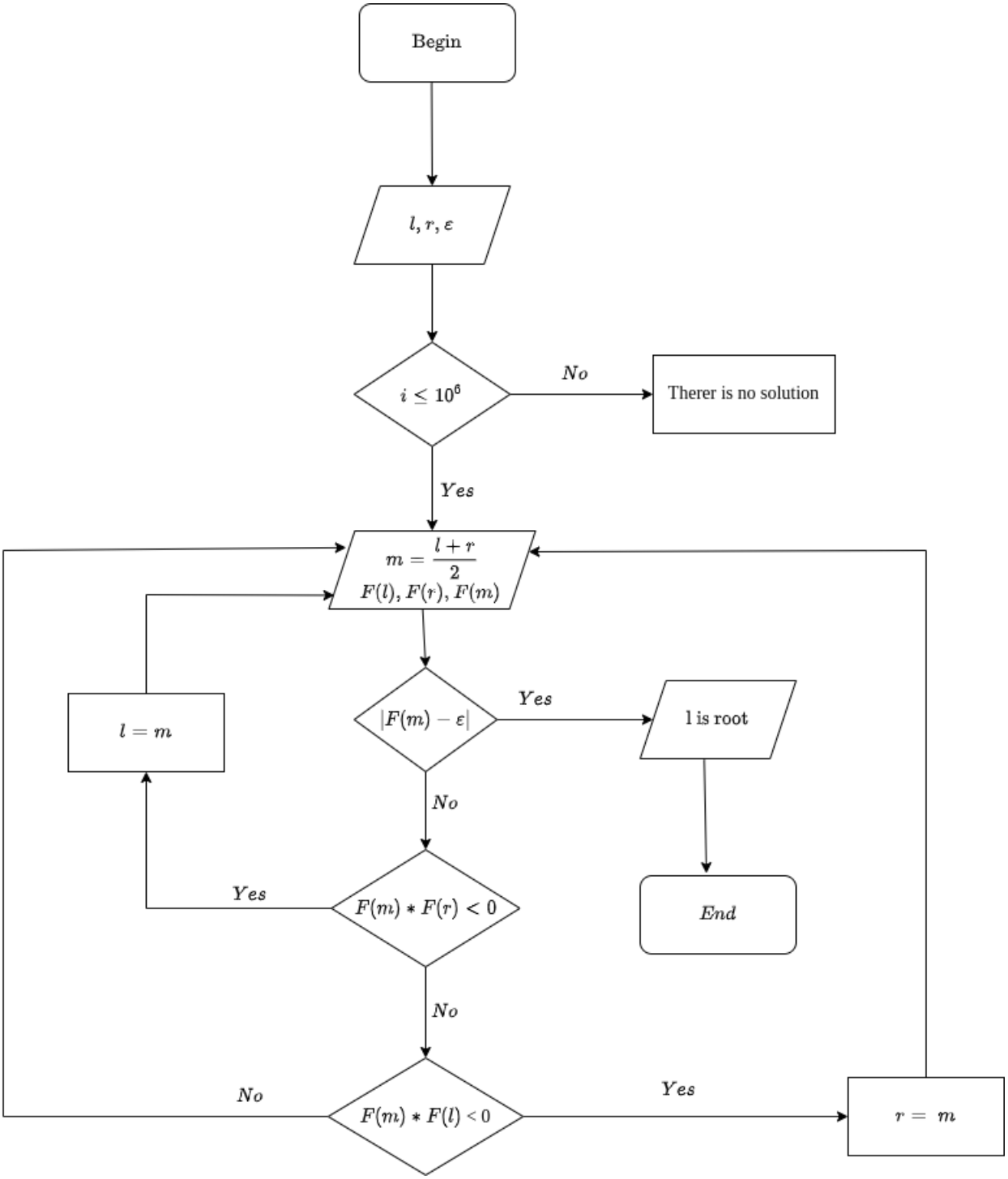
Исходное уравнение заменим следующим $F(x^0) + F(x^0)(x - x^0) = 0$. Считая матрицу Якоби $F(x^0)$ неособой, разрешим это уравнение относительно x: $x = x^0[F(x)]^{-1}F(x^0)$. И вообще положим:

$$x^{k+1} = x^k - [F(x^k)]^{-1} F(x^k)$$

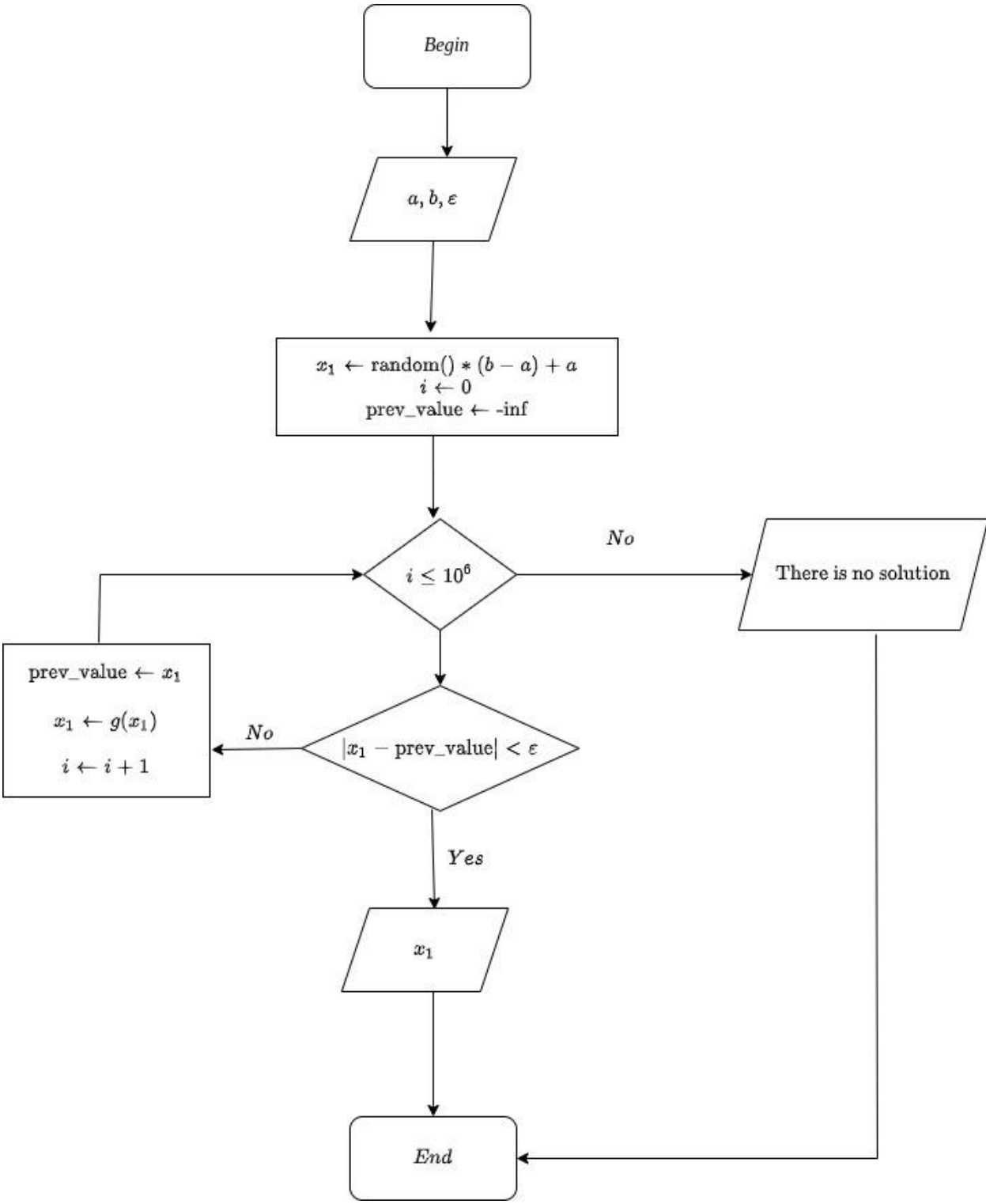
II. Блок схемы

A. Решение нелинейных уравнений

1. Метод деления пополам

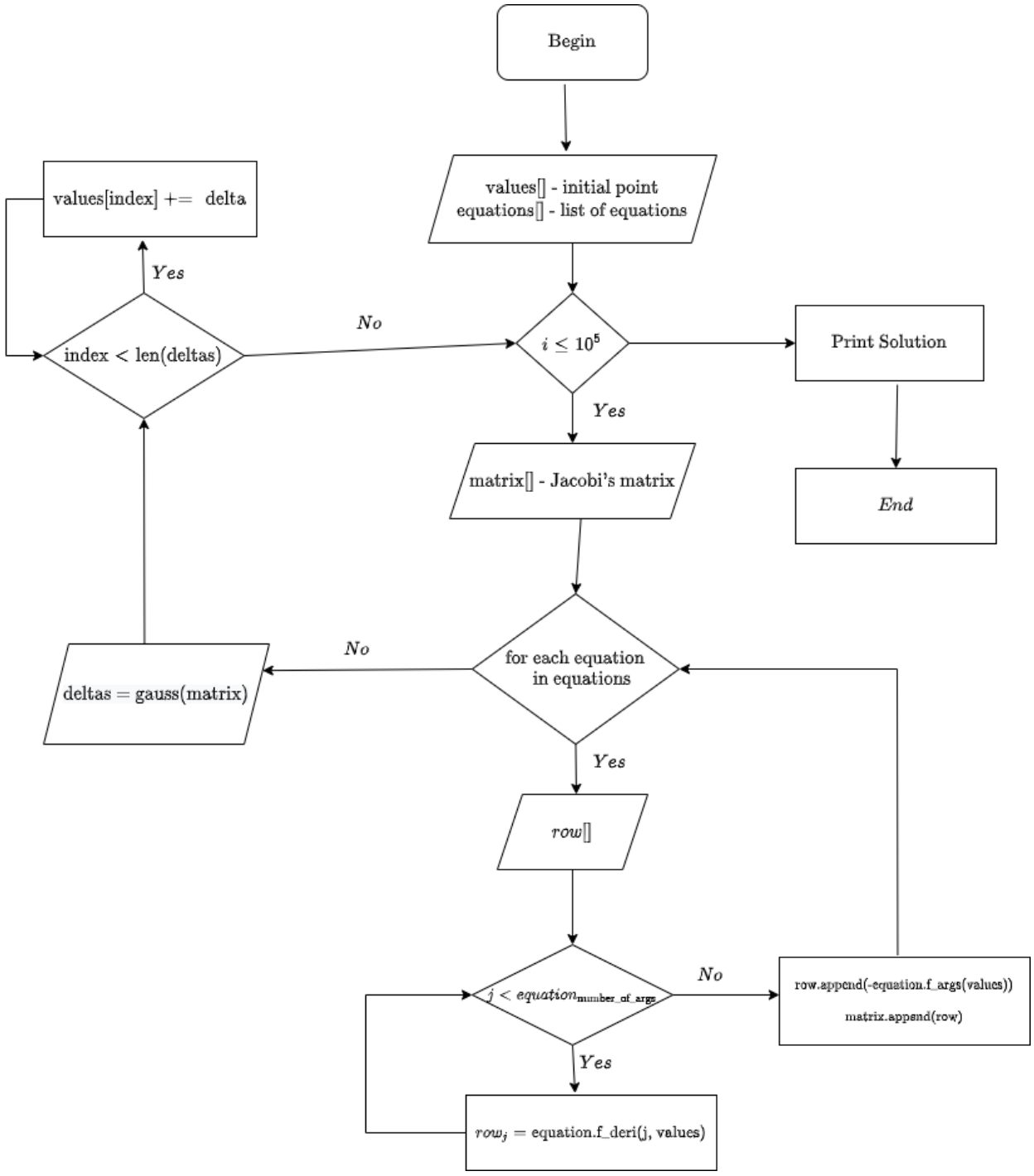


2. Метод простой итерации



B. Решение систем нелинейных уравнений

1. Метод Ньютона



III. Листинги

A. Решение нелинейных уравнений

1. Метод деления пополам

```
def find_root(self):
    print("> Please enter your interval [a, b]: ", end = "")
    l, r = map(float, input().split())
    print("> Please enter your epsilon: ", end = "")
    self.EPS = float(input())

    for i in range(int(1e6)):
        m = (l + r) / 2
        fl = self.nonlinear_equation.f(l)
        fr = self.nonlinear_equation.f(r)
        fm = self.nonlinear_equation.f(m)
        if abs(fm) < self.EPS:
            l = m
            break
        if fm * fr < 0: l = m
        elif fl * fm < 0: r = m

    return l if abs(self.nonlinear_equation.f(l)) < self.EPS else "There is no solution!"
```

2. Метод простой итерации

```
def find_root(self):
    print("> please enter your interval [a, b]: ", end = "")
    a, b = map(float, input().split())
    print("> Please enter your epsilon: ", end = "")
    self.EPS = float(input())
    x1 = random.random() * (b - a) + a
    prv_val = -1e9
    iter = 0
    while iter < int(1e6):
        iter += 1
        if abs(self.__nonlinear_equation.f(x1)) < self.EPS and abs(prv_val - x1) < self.EPS:
            return x1
        if (x1.real < a or x1.real > b): return "There is no solution!!"
        x1 = self.__nonlinear_equation.f_equiv(x1)


    return x1 if abs(self.__nonlinear_equation.f(x1)) < self.EPS else "There is no solution!"
```

B. Решение систем нелинейных уравнений

1. **Метод Ньютона**

```python
class NewtonMethod(Method):
    size = 0
    SCALE = 6
    def execute(self):
        equations = self.system.get_equations()
        for equa in equations:
            self.size = max(self.size, equa.get_number_of_args())
        print("> Please enter an initial value of {} arguments: ".format(self.size))
        print("> ", end = "")

        values = list(map(float, input().split()))

        for i in range(int(1e5)):
            matrix = []
            for equation in equations:
                row = []
                for j in range(equation.get_number_of_args()):
                    row.append(equation.f_deri(j, values))
                row.append(-equation.f_args(values))
                matrix.append(row)
            deltas = self.gauss(matrix)
            for index, delta in enumerate(deltas): values[index] += delta


        return values
```

```python
  def gauss(self, matrix):
      n = len(matrix)
      pivot_col, pivot_row = 0, 0
      while pivot_col < n and pivot_row < n:
          i = pivot_row
          for i in range(pivot_row, n, 1):
              if matrix[i][pivot_col]: break
          if not matrix[i][pivot_col]:
              pivot_col += 1
              continue
          matrix[i], matrix[pivot_row] = matrix[pivot_row], matrix[i]
          for i in range(pivot_row + 1, n, 1):
              rate = matrix[i][pivot_col] / matrix[pivot_row][pivot_col]
              for j in range(pivot_col, n + 1, 1):
                  matrix[i][j] -= matrix[pivot_row][j] * rate

          pivot_row += 1
          pivot_col += 1
      values = []
      for i in range(n - 1, -1, -1):
          for j in range(n - 1, i, -1):
              matrix[i][n] -= values[n-1-j] * matrix[i][j]
          values.append(matrix[i][n] / matrix[i][i])
      return reversed(values)
```

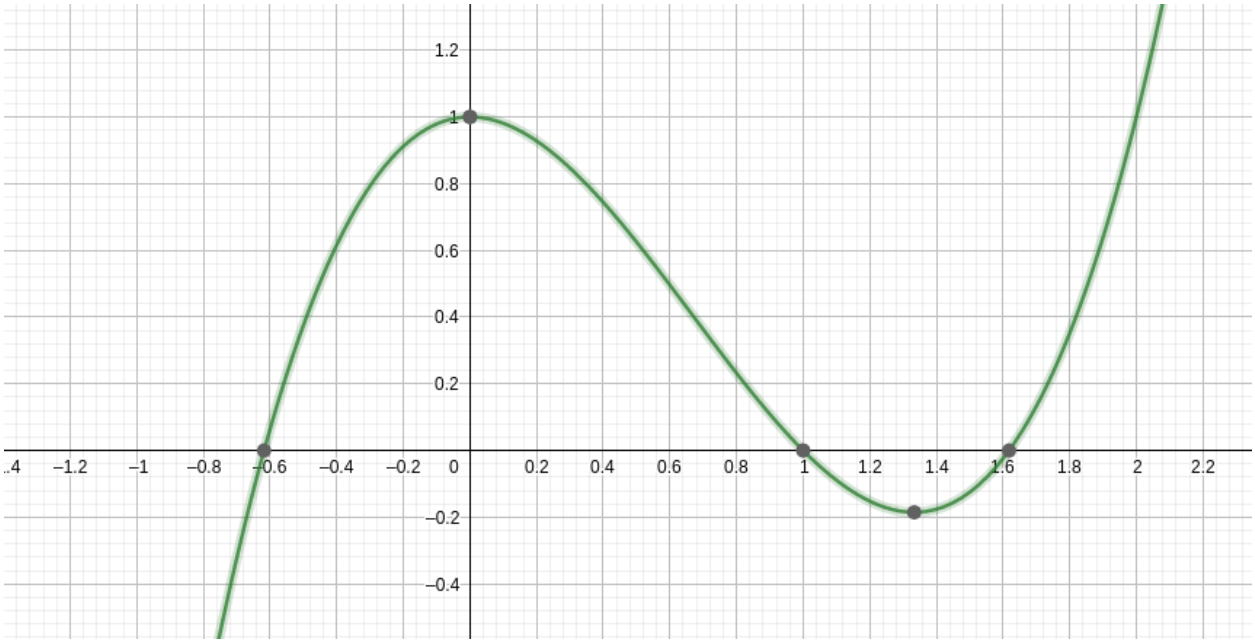
IV. Примеры и результат работы программы

A. Решение нелинейных уравнений

- **Пример 1:**

$$x^3 - 2x^2 + 1 = y$$

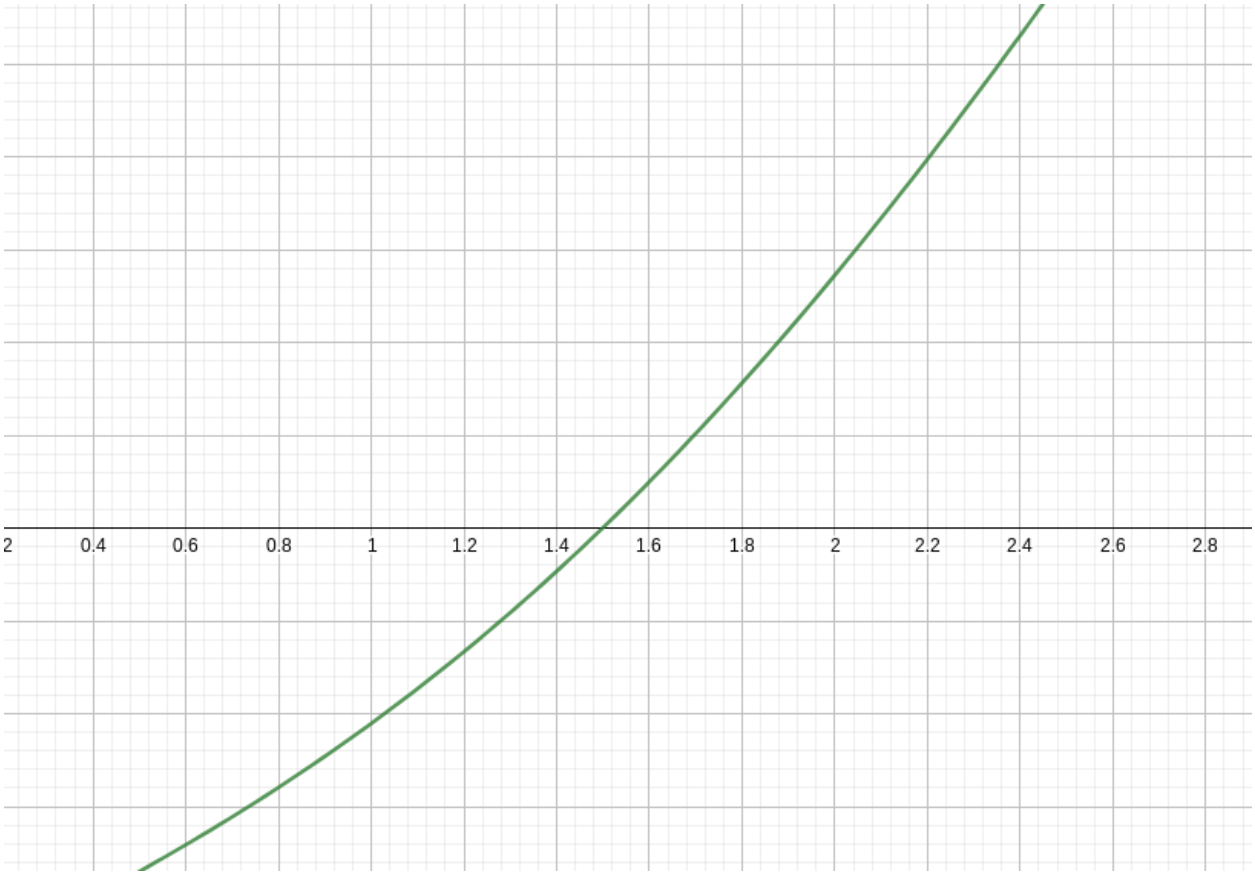
```
[ndwannafly@ndwannafly-hnwx9x comp_math_lab_2]$ python3 main.py
> ---------- Welcome to nonlinear world! ----------
> Which type of problem do you want to solve?
> 1. Solve a nonlinear equation.
> 2. Solve a system of nonlinear equations
> Enter your option (1 or 2): 1
> Please choose one of the equations below
> 0. x^3 - 2x^2 + 1 = y
> 1. x - 1 - 0.5 * sin(x) = y
> 2. ln(x) + x^3 - x = y
> Enter your choice: 0
> ---------- SIMPLE ITERATION METHOD ----------
> please enter your interval [a, b]: -5 5
> Please enter your epsilon: 0.01
> The solution of this equation is: 1.618034
> ---------- BISECTION METHOD ----------
> Please enter your interval [a, b]: -5 5
> Please enter your epsilon: 0.01
> The solution of this equation is: -0.620117
[ndwannafly@ndwannafly-hnwx9x comp_math_lab_2]$
```



- **Пример 2:**

$$x - 1 - 0.5 * sin(x) = y$$

```
[ndwannafly@ndwannafly-hnwx9x comp_math_lab_2]$ python3 main.py
> ---------- Welcome to nonlinear world! ----------
> Which type of problem do you want to solve?
> 1. Solve a nonlinear equation.
> 2. Solve a system of nonlinear equations
> Enter your option (1 or 2): 1
> Please choose one of the equations below
> 0. x^3 - 2x^2 + 1 = y
> 1. x - 1 - 0.5 * sin(x) = y
> 2. ln(x) + x^3 - x = y
> Enter your choice: 1
> ---------- SIMPLE ITERATION METHOD ----------
> please enter your interval [a, b]: -5 5
> Please enter your epsilon: 0.01
> The solution of this equation is: 1.498701
> ---------- BISECTION METHOD ----------
> Please enter your interval [a, b]: -5 5
> Please enter your epsilon: 0.01
> The solution of this equation is: 1.503906
[ndwannafly@ndwannafly-hnwx9x comp_math_lab_2]$
```



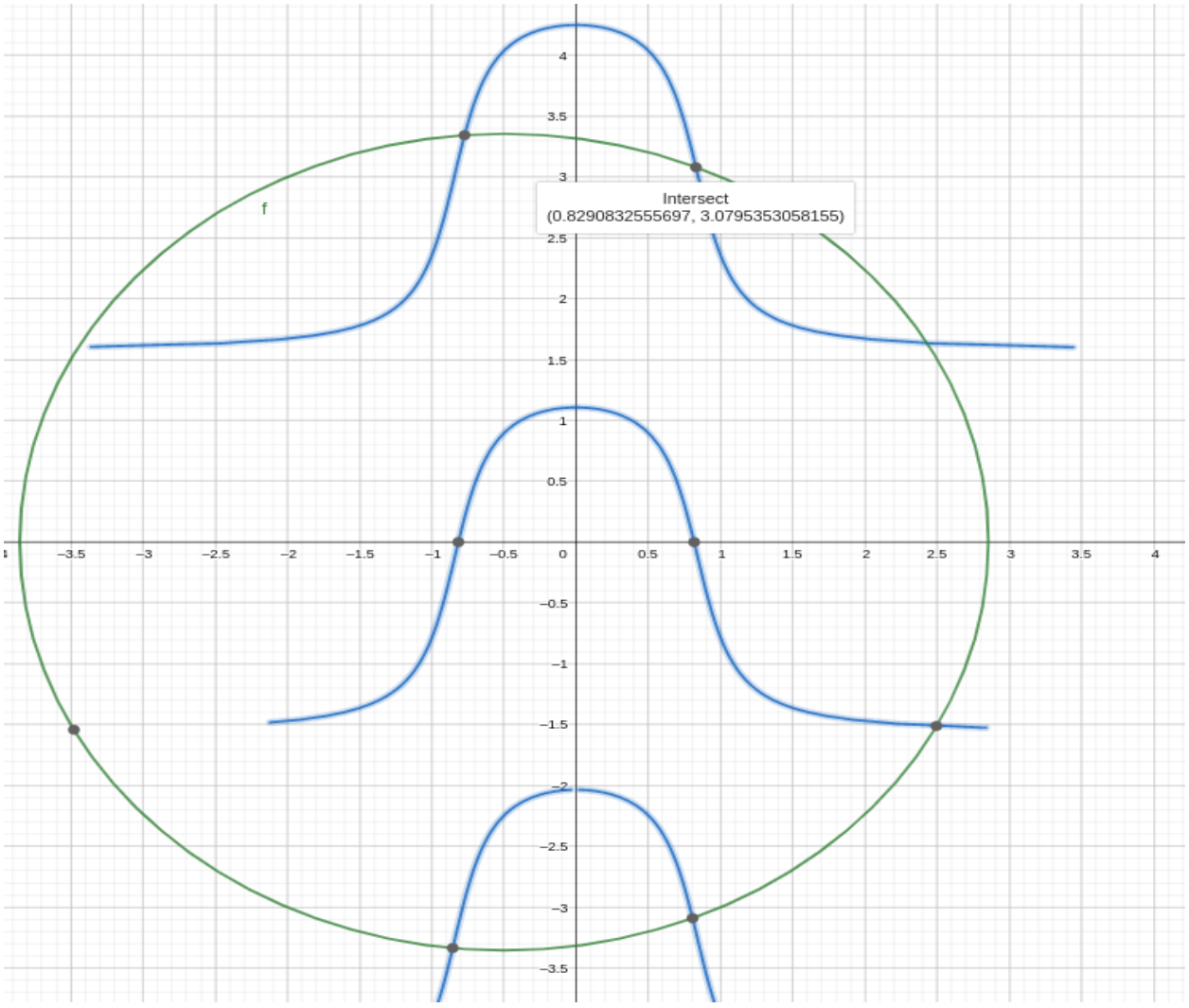
B. Решение систем нелинейных уравнений

- **Пример 1:**

$$x^2 + x - 11 + y^2 = 0$$

$$3x^2 + tan(y) - 2 = 0$$

```
[ndwannafly@ndwannafly-hnwx9x comp_math_lab_2]$ python3 main.py
> ---------- Welcome to nonlinear world! ----------
> Which type of problem do you want to solve?
> 1. Solve a nonlinear equation.
> 2. Solve a system of nonlinear equations
> Enter your option (1 or 2): 2
> Please choose one of the system of nonlinear equations below:
> 0. System 0
   3 * x^2 - 5 + 2 * y^2 = 0
   8 * x^2 - 14 + 6 * y^2 = 0
> 1. System 1
   x^2 + x - 11 + y^2 = 0
   3 * x^2 + tan(y) - 2 = 0
> 2. System 2
   7 * x^3 - 2 + -6 * y^2 + y = 0
   x^4 - 5 + y^3 = 0
> Enter your variant: 1
> Please enter an initial value of 2 arguments:
> 2 2
The solution of this system of nonlinear equations is: 0.829 3.08
[ndwannafly@ndwannafly-hnwx9x comp_math_lab_2]$
```



V. Вывод

В результате выполнения я изучил 4 метода решения нелинейных уравнений и 2 метода решения систем нелинейных уравнений, пришел к следующим выводам относительно их преимуществ и недостатков:

A. Методы решения нелинейных уравнений

1. **Метод касательных:**

- Достоинства: Метод обладает квадратичной сходимостью
- Недостатки:
    + Необходимость вычисления производной на каждой итерации
    + Нет гарантии, что функция должна сходиться

2. **Метод простой итерации:**

- Достоинства: Простота реализации
- Недостатки:
    + Сходимость метода в малой окрестности корня и вытекающая отсюда необходимость выбора начального приближения к корню из этой малой окрестности.
    + В противном случае итерационный процесс расходится или сходится к другому корню этого уравнения. Также при $|\varphi(x)| \approx 1$, то сходимость может быть очень медленной

3. **Метод половинного деления:**

- Достоинства:
    + Имеет смысл применять в случаях когда требуется высокая надежность счета, а скорость несущественна.
    + Обладает абсолютной сходимостью (близость получаемого численного решения задачи к истинному решению.) Устойчив к ошибкам округления.
- Недостатки:
    + если интервал содержит несколько корней, то неизвестно к какому относится вычислительный процесс.
    + Иногда бывает непросто найти две точки, знаки которых различны.

4. **Метод хорд:**

- Достоинства:
    + Простота реализации.
    + Нет необходимости вычисления производной на каждой итерации.
    + Работает, даже если мы не знаем точный интервал.
- Недостатки:
    + Скорость сходимости - линейная.
    + Порядок сходимости метода хорд выше, чем у метода половинного деления.

В. Методы решения систем нелинейных уравнений

1. **Метод Ньютона:**
   Метод Ньютона для решения СНАУ представляет собой обобщение метода Ньютона для решения НУ его сутью является попытка свести решение системы нелинейных уравнений к решению системы линейных уравнений. Основная сложность метода Ньютона заключается в обращении матрицы Якоби. Вводя обозначение $\Delta x^{(k)} = x^{(k+1)} - x^{(k)} \Delta x^{(k)} = x^{(k+1)} - x^{(k)}$ получаем СЛАУ для вычисления $\Delta x^{(k)}$. Решение этого СЛАУ создает основную вычислительную нагрузку алгоритма.
2. **Метод простой итерации:**

- Метод простой итераций, в свою очередь позволяет грубо говоря подобрать вектор решений системы уравнений путем выражения значения одной неизвестной через все остальные и постепенной подстановки значений неизвестных, вычисляемых на каждом шаге.