



$$y = mx + b$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

# NUMERICAL METHODS

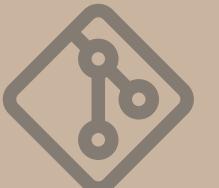
---

## IMPLEMENTATION IN C++

A Structured Repository Approach

$$ax^2 + bx + c = 0$$

$$= x + x^{\frac{i}{n}}$$



# TEAM MEMBERS

1

**MD. TAKI TAHMID SAAD**  
**2207022**  
**Second Year**  
**Second Term**

2

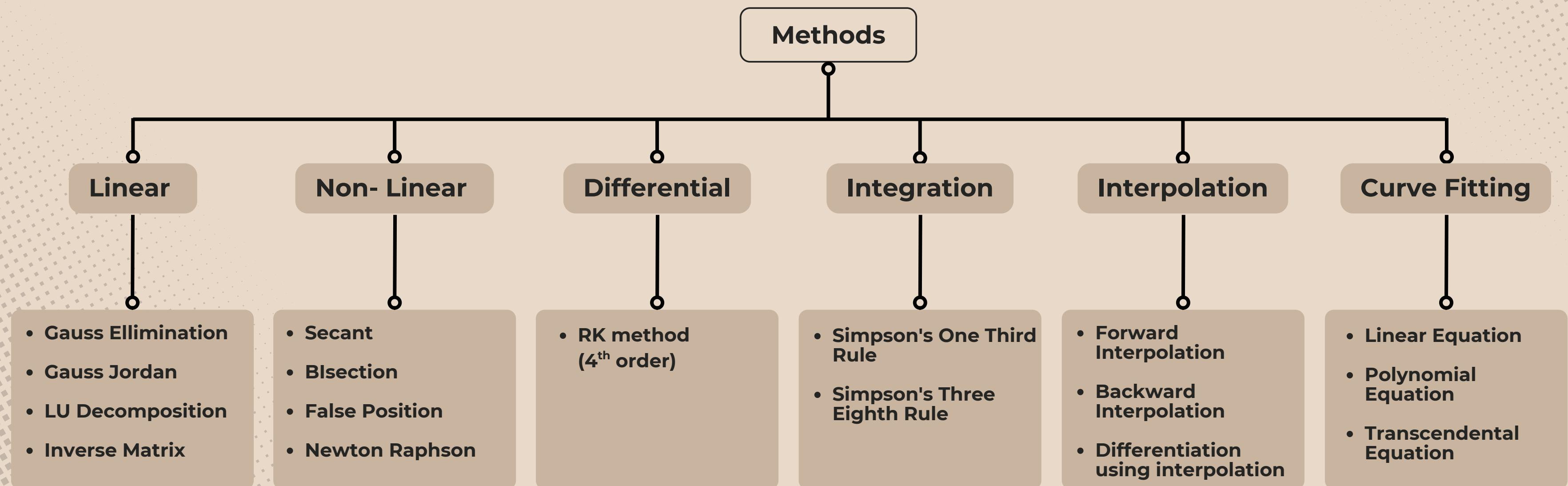
**SALEH SADID MIR**  
**2207024**  
**Second Year**  
**Second Term**

3

**MD SADIKUL ISLAM SIYAM**  
**2207031**  
**Second Year**  
**Second Term**

# PROJECT AT A GLANCE

A collection of numerical methods implemented in C++ for solving various mathematical problems including linear equations, non-linear equations, differential equations, integration, interpolation, and curve fitting.



# MAIN INTERFACE

A table-based navigation system linking 17 numerical methods to their theory, code, input files, and output files.

The screenshot shows a user interface for a "Numerical Methods Project". At the top left is a "README" button with a file icon. On the right are edit and list icons. The main title "Numerical Methods Project" is displayed prominently. Below it is a descriptive paragraph: "A collection of numerical methods implemented in C++ for solving various mathematical problems including linear equations, non-linear equations, interpolation, integration, differential equations, and curve fitting." A horizontal line separates this from the "Table of Contents". The "Table of Contents" section lists six categories: Non-Linear Equations, Linear Equations, Interpolation Methods, Integration Methods, Differential Equations, and Curve Fitting - Regression. Each category is preceded by a numbered link (1. through 6.) and underlined text.

**README**

## Numerical Methods Project

A collection of numerical methods implemented in C++ for solving various mathematical problems including linear equations, non-linear equations, interpolation, integration, differential equations, and curve fitting.

---

### Table of Contents

1. [Non-Linear Equations](#)
2. [Linear Equations](#)
3. [Interpolation Methods](#)
4. [Integration Methods](#)
5. [Differential Equations](#)
6. [Curve Fitting - Regression](#)

# PROJECT STRUCTURE

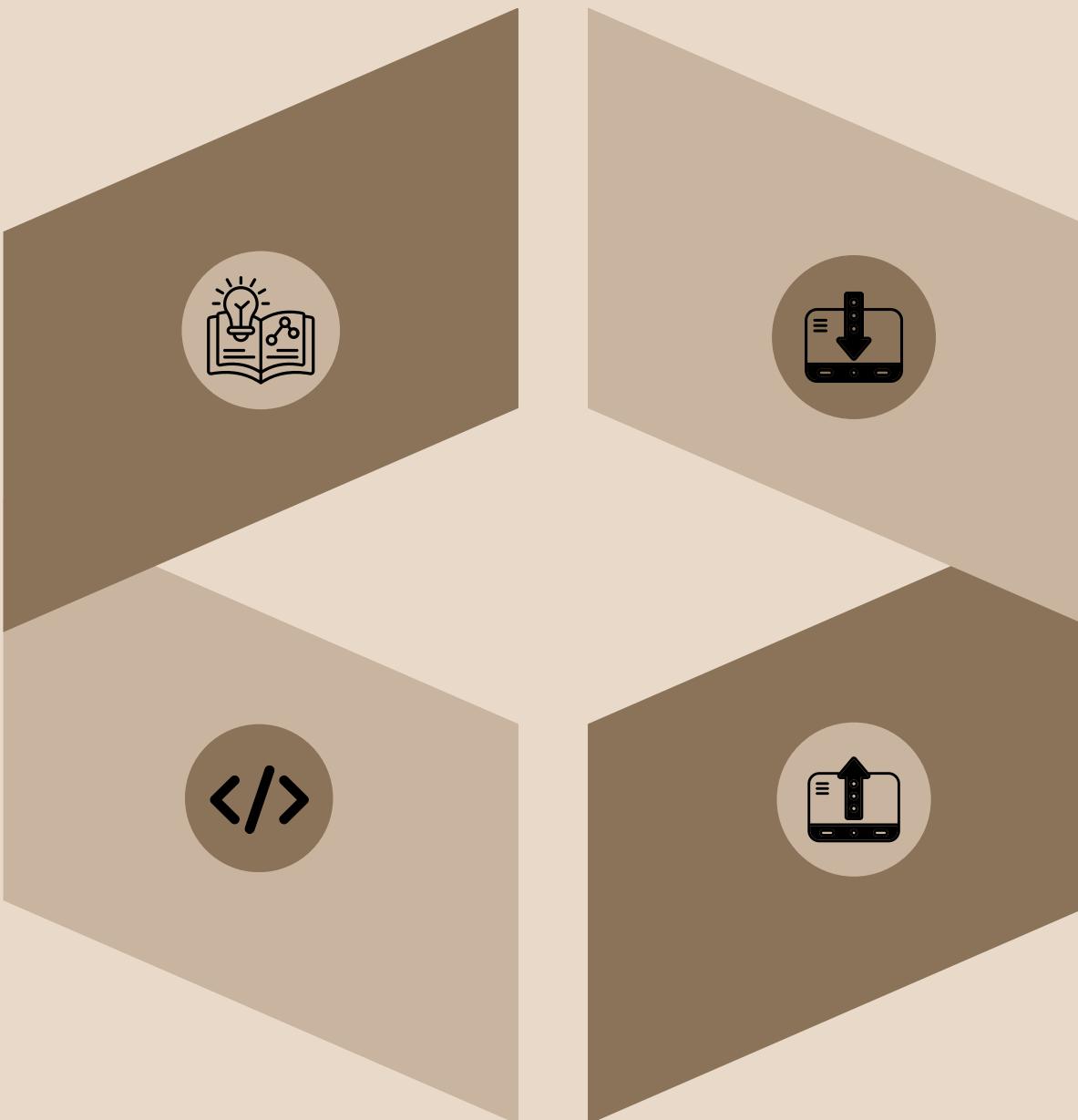
Each method folder contains

## THEORY

A README file containing the theory, equation , algorithm and code constraint of the method

## CODE

Contains the code for the respective method



## INPUT FILE

Handles the input for the respective C++ code

## OUTPUT FILE

Handles the output of the respective C++ code

Methods for solving systems of linear equations ( $AX = B$ ).

| Method                   | Theory               | Code                 | Input                | Output               |
|--------------------------|----------------------|----------------------|----------------------|----------------------|
| Gauss Elimination Method | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |
| Gauss Jordan Method      | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |
| LU Decomposition Method  | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |
| Inverse Matrix Method    | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |

# LINEAR EQUATIONS

# NON-LINEAR EQUATIONS

Methods for finding roots of equations of the form  $f(x) = 0$ .

| Method                | Theory               | Code                 | Input                | Output               |
|-----------------------|----------------------|----------------------|----------------------|----------------------|
| Bisection Method      | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |
| False Position Method | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |
| Newton Raphson Method | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |
| Secant Method         | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |

# INTERPOLATION

Methods for estimating values between known data points.

| Method                                       | Theory               | Code                 | Input                | Output               |
|--|----------------------|----------------------|----------------------|----------------------|
| Newton's Forward & Backward Interpolation    | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |
| Differentiation using Newton's Interpolation | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |

Least square regression methods for fitting curves to experimental data.

| Type                    | Theory               | Code                 | Input                | Output               |
|-------------------------|----------------------|----------------------|----------------------|----------------------|
| Linear Equation         | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |
| Polynomial Equation     | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |
| Transcendental Equation | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |

# CURVE FITTING

# INTEGRATION

Methods for solving ordinary differential equations (ODEs).

| Method                   | Theory               | Code                 | Input                | Output               |
|--------------------------|----------------------|----------------------|----------------------|----------------------|
| Runge-Kutta Method (RK4) | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |

# DIFFERENTIAL EQUATIONS

Numerical methods for finding definite integrals.

| Method             | Theory               | Code                 | Input                | Output               |
|--------------------|----------------------|----------------------|----------------------|----------------------|
| Simpson's 1/3 Rule | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |
| Simpson's 3/8 Rule | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> | <a href="#">View</a> |

# THEORY

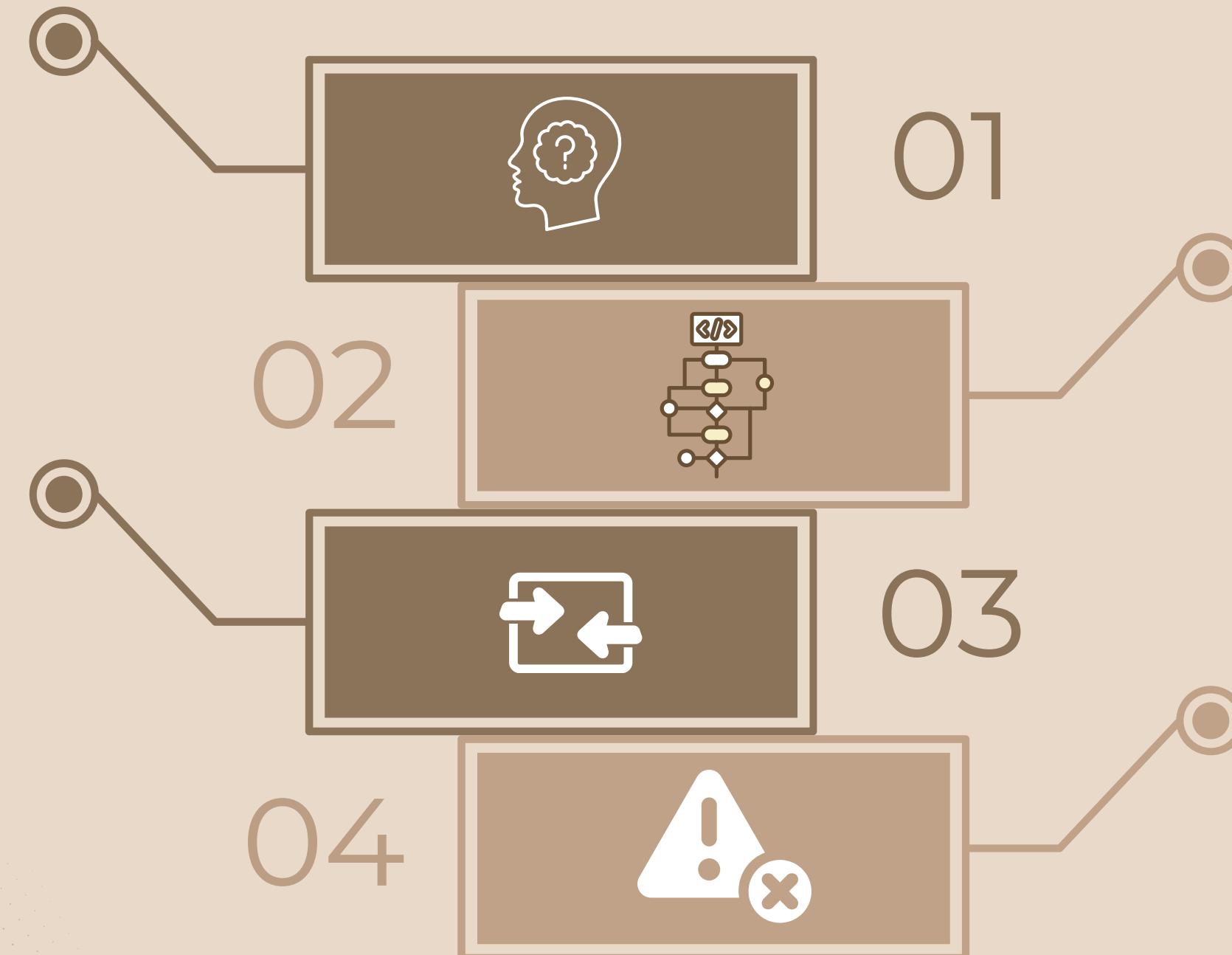
Each README file under each folder of each method contains

## INTRODUCTION

The mathematical foundation and formulas behind the method

## I/O EXAMPLE

Sample Input and output for the method



## ALGORITHM

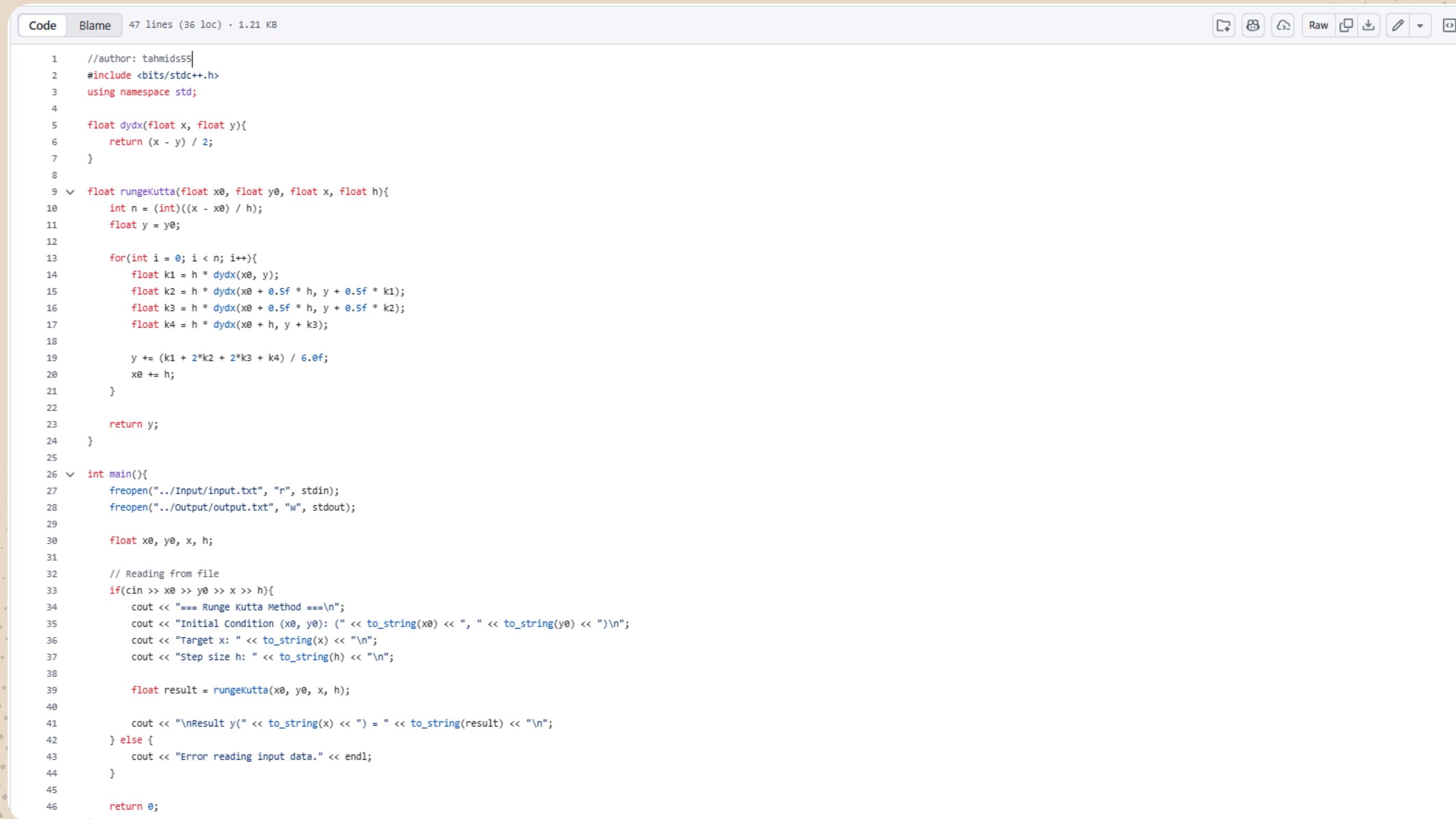
Step-by-step procedure to implement the method

## CODE CONSTRAINTS

Conditions where the method works or fails

# CODE SECTION

Each code.cpp file contains the C++ implementation of the respective method



The screenshot shows a code editor window with the following details:

- Tab Bar:** Code (selected), Blame, 47 lines (36 loc) · 1.21 KB.
- Toolbar:** Includes icons for copy, paste, find, search, raw view, and download.
- Code Area:** Displays the following C++ code:

```
1 //author: tahmidss5
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 float dydx(float x, float y){
6     return (x - y) / 2;
7 }
8
9 float rungeKutta(float x0, float y0, float x, float h){
10    int n = (int)((x - x0) / h);
11    float y = y0;
12
13    for(int i = 0; i < n; i++){
14        float k1 = h * dydx(x0, y);
15        float k2 = h * dydx(x0 + 0.5f * h, y + 0.5f * k1);
16        float k3 = h * dydx(x0 + 0.5f * h, y + 0.5f * k2);
17        float k4 = h * dydx(x0 + h, y + k3);
18
19        y += (k1 + 2*k2 + 2*k3 + k4) / 6.0f;
20        x0 += h;
21    }
22
23    return y;
24 }
25
26 int main(){
27     freopen("../Input/input.txt", "r", stdin);
28     freopen("../Output/output.txt", "w", stdout);
29
30     float x0, y0, x, h;
31
32     // Reading from file
33     if(cin >> x0 >> y0 >> x >> h){
34         cout << "Runge Kutta Method\n";
35         cout << "Initial Condition (x0, y0): (" << to_string(x0) << ", " << to_string(y0) << ")\n";
36         cout << "Target x: " << to_string(x) << "\n";
37         cout << "Step size h: " << to_string(h) << "\n";
38
39         float result = rungeKutta(x0, y0, x, h);
40
41         cout << "Result y(" << to_string(x) << ") = " << to_string(result) << "\n";
42     } else {
43         cout << "Error reading input data." << endl;
44     }
45
46     return 0;
47 }
```

**Fig 01:** Code for Runge-Kutta method

# I/O HANDLING

**input.txt**

```
3  
2 1 -1 8  
-3 -1 2 -11  
-2 1 2 -3
```

**output.txt**

Given System of Equations:

$$(2x1) + (1x2) + (-1x3) = 8$$

$$(-3x1) + (-1x2) + (2x3) = -11$$

$$(-2x1) + (1x2) + (2x3) = -3$$

Solution Type: UNIQUE SOLUTION

Solution:

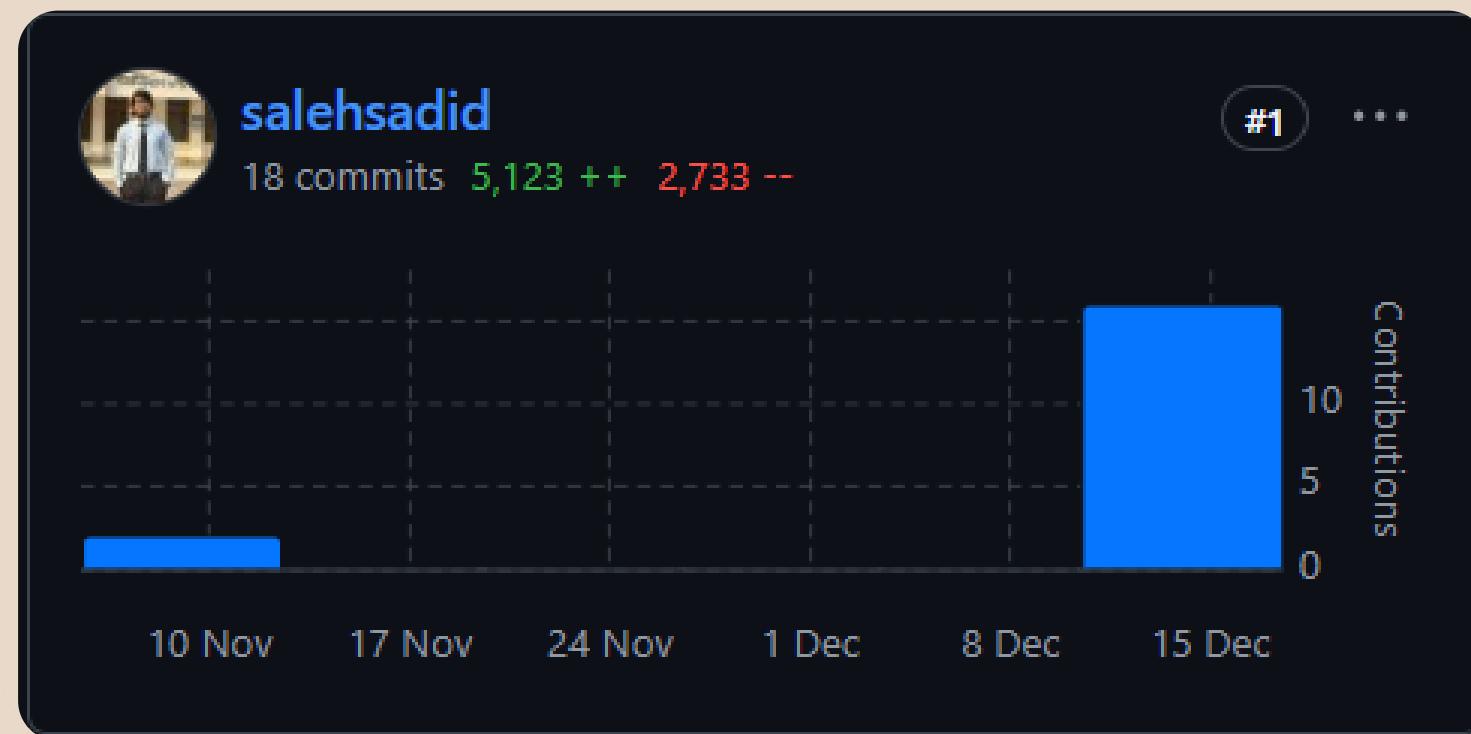
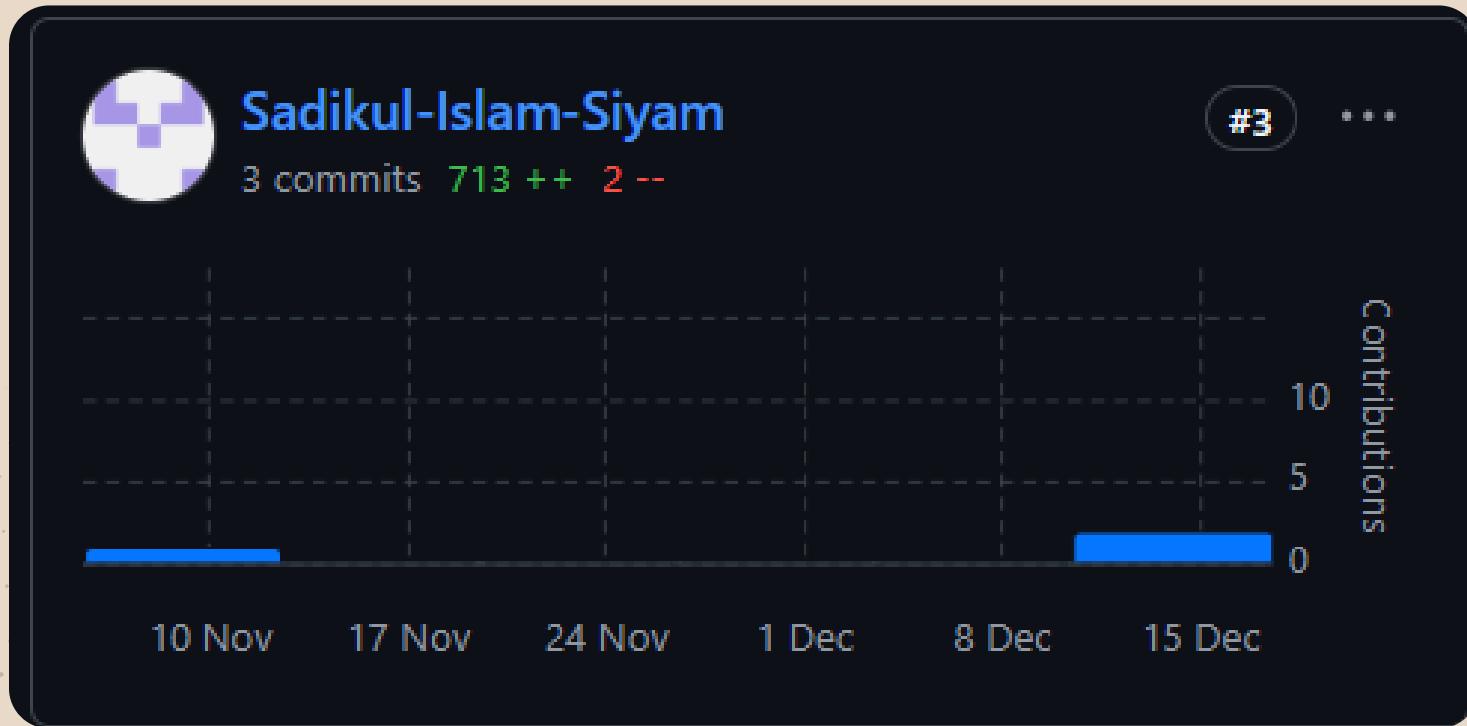
$$x1 = 2.000000$$

$$x2 = 3.000000$$

$$x3 = -1.000000$$

**Fig 02:** I/O files for Gauss Jordan Method

# GITHUB COLLABORATION



$$y = mx + b$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

# — THANK YOU —

GitHub Repository Link

🔗 [\*\*https://github.com/salehsadid/Numeric-Methods-Project\*\*](https://github.com/salehsadid/Numeric-Methods-Project)

$$ax^2 + bx + c = 0$$

$$= \times \times \div$$

SCAN ME

