CSE 2208

$y = mx + b$

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

NUMERICAL METHODS

IMPLEMENTATION IN C++

A Structured Repository Approach

$ax^2 + bx + c = 0$

$= + \times \div -$

TEAM MEMBERS

1

MD. TAKI TAHMID SAAD
2207022
Second Year
Second Term

2

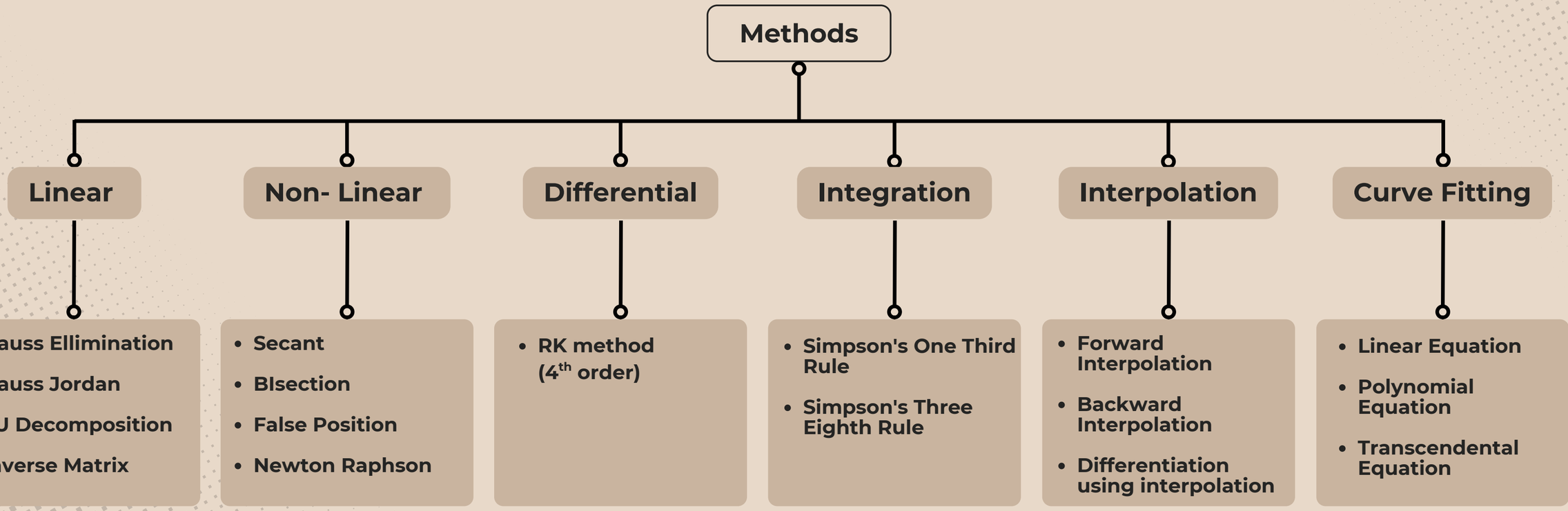
SALEH SADID MIR
2207024
Second Year
Second Term

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.

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

PROJECT STRUCTURE

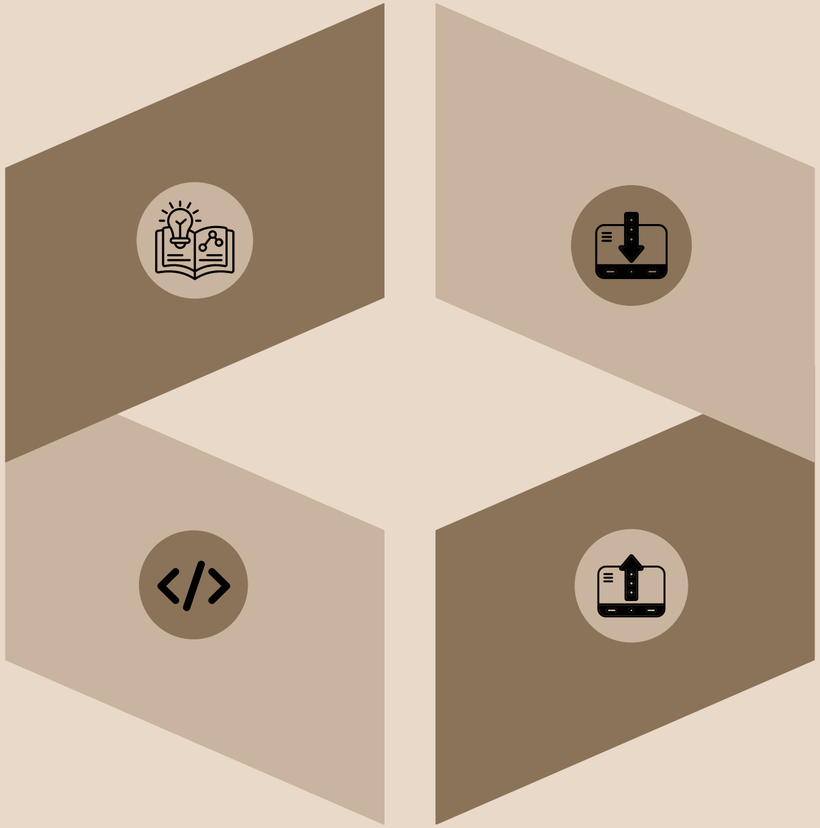
Each method folder contains

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

LINEAR EQUATIONS

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

Method	Theory	Code	Input	Output
Gauss Elimination Method	View	View	View	View
Gauss Jordan Method	View	View	View	View
LU Decomposition Method	View	View	View	View
Inverse Matrix Method	View	View	View	View

NON-LINEAR EQUATIONS

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

Method	Theory	Code	Input	Output
Bisection Method	View	View	View	View
False Position Method	View	View	View	View
Newton Raphson Method	View	View	View	View
Secant Method	View	View	View	View

INTERPOLATION

Methods for estimating values between known data points.

Method	Theory	Code	Input	Output
Newton's Forward & Backward Interpolation	View	View	View	View
Differentiation using Newton's Interpolation	View	View	View	View

CURVE FITTING

Least square regression methods for fitting curves to experimental data.

Type	Theory	Code	Input	Output
Linear Equation	View	View	View	View
Polynomial Equation	View	View	View	View
Transcendental Equation	View	View	View	View

DIFFERNTIAL EQUATIONS

Methods for solving ordinary differential equations (ODEs).

Method	Theory	Code	Input	Output
Runge-Kutta Method (RK4)	View	View	View	View

INTEGRATION

Numerical methods for finding definite integrals.

Method	Theory	Code	Input	Output
Simpson's 1/3 Rule	View	View	View	View
Simpson's 3/8 Rule	View	View	View	View

THEORY

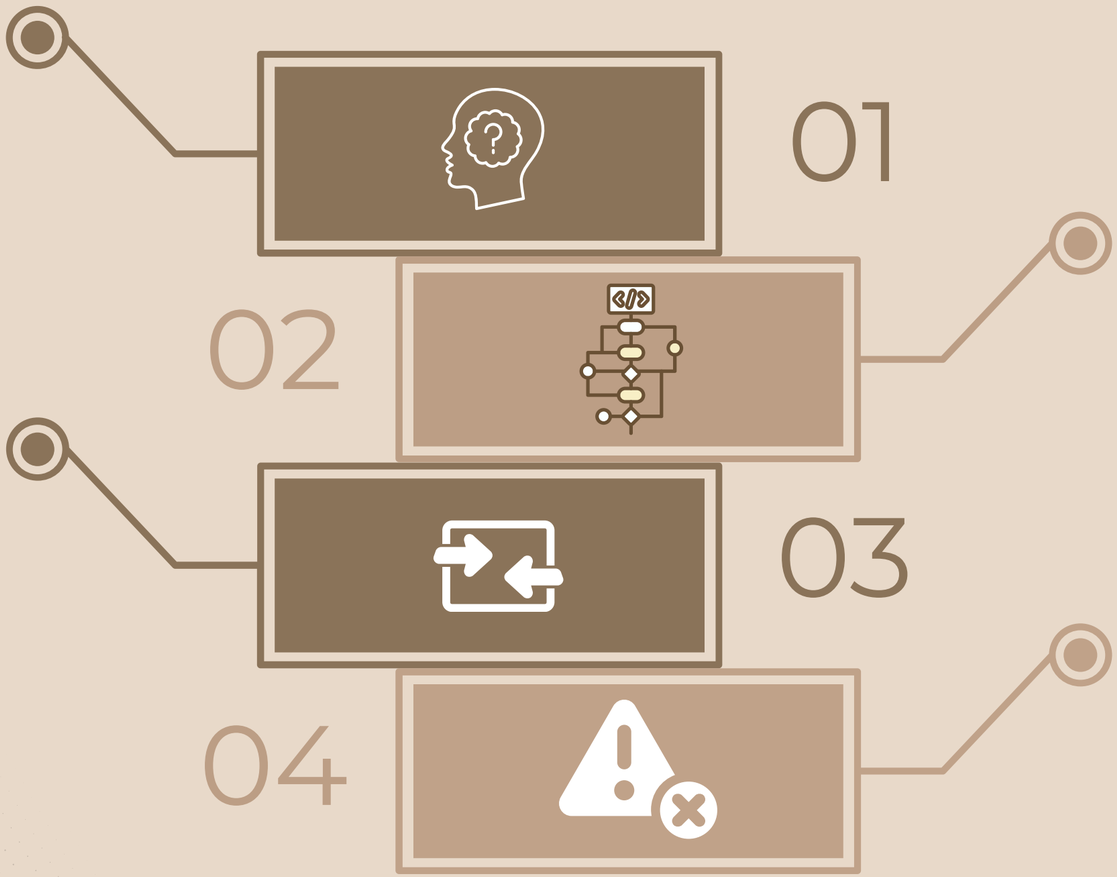
Each README file under each folder of each method contains

INTRODUCTION

The mathematical foundation and formulas behind the method

I/O EAXMPLE

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

Code Blame 47 lines (36 loc) · 1.21 KB Raw

```cpp
//author: tahmids55
#include <bits/stdc++.h>
using namespace std;

float dydx(float x, float y){
    return (x - y) / 2;
}

float rungeKutta(float x0, float y0, float x, float h){
    int n = (int)((x - x0) / h);
    float y = y0;

    for(int i = 0; i < n; i++){
        float k1 = h * dydx(x0, y);
        float k2 = h * dydx(x0 + 0.5f * h, y + 0.5f * k1);
        float k3 = h * dydx(x0 + 0.5f * h, y + 0.5f * k2);
        float k4 = h * dydx(x0 + h, y + k3);

        y += (k1 + 2*k2 + 2*k3 + k4) / 6.0f;
        x0 += h;
    }

    return y;
}

int main(){
    freopen("../Input/input.txt", "r", stdin);
    freopen("../Output/output.txt", "w", stdout);

    float x0, y0, x, h;

    // Reading from file
    if(cin >> x0 >> y0 >> x >> h){
        cout << "=== Runge Kutta Method ===\n";
        cout << "Initial Condition (x0, y0): (" << to_string(x0) << ", " << to_string(y0) << ")\n";
        cout << "Target x: " << to_string(x) << "\n";
        cout << "Step size h: " << to_string(h) << "\n";

        float result = rungeKutta(x0, y0, x, h);

        cout << "\nResult y(" << to_string(x) << ") = " << to_string(result) << "\n";
    } else {
        cout << "Error reading input data." << endl;
    }

    return 0;
```

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

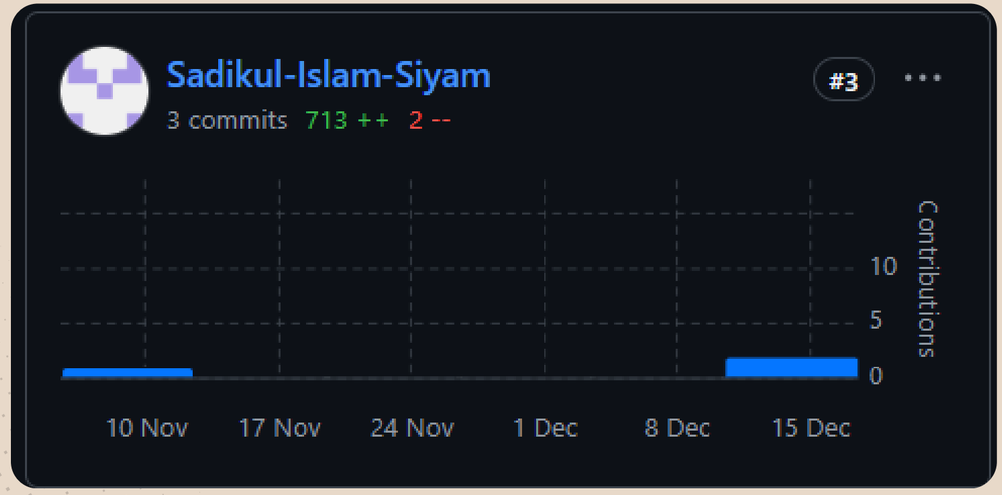
```
Given System of Equations:
(2x1) + (1x2) + (-1x3) = 8
(-3x1) + (-1x2) + (2x3) = -11
(-2x1) + (1x2) + (2x3) = -3

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

$ax^2 + bx + c = 0$

SCAN ME