190503

MTH2008

Scientific computing

Second semester

4th lecture

Danilo Roccatano - Marco Pinna

Office : INB3129

Email: Mpinna@lincoln.ac.uk

Padlet

https://uol.padlet.org/mpinna2/scientific-computing-l5c5cjv5fmvv72l0

Logbook

- The logbook structure will follow the same structure as previously explained in the first semester
- Only the part of the second semester topics should be sumbitted.

Notes

For each lecture, I have created a program folder containing all the relevant code. Please avoid cutting and pasting directly from the PowerPoint slides, as this may result in errors (not always). Instead, ensure you copy the code directly from the program folder.

Overview Semester B

NUMERICAL METHODS AND ALGORITHMS

- Random Numbers
- Sorting algorithm
- Numerical Differentiation
- Root of a equations
  - Bisection Method
  - Newton Method
  - Secant Method
- Numerical Integration
- Numerical integration of Ordinary Differential Equation (first order)
- Linear Algebra
  - Product of scalar with vectors/matrix
  - Product of two matrix

C++ Coding

- Open, read and write a file in C++
- Dynamical arrays

Solution of task 3.1

Task 3.1: Matrix multiplication, File I/O and matrix manipulation in C++

1.Generate Matrix A and B:

- The user has to enter the dimensions for matrices A and B.
- Automatically generate random integers (between 1 and 100) to fill both matrices
- Ensure that the number of columns in Matrix A equals the number of rows in Matrix B to allow the matrix multiplication

2.Matrix multiplication

- Multiply matrices A and B to produce matrix C.
- Display all the three Matrices (A, B, C) in the console in a readable format

3. Save the Matrix to a File:

- Write the generated matrices to a file named `matrix_multiplication.txt`
- Format the output so each matrix is clearly labeled and readable.

4.Error handling:

- Ensure the program gracefully handles cases where matrix multiplication is not possible (e.g., dimension mismatch).

- Use `rand()` and `srand()` for random number generation (or modern generator)
- Utilize file handling with `ofstream` (for writing) and `ifstream` (for reading).
- Ensure the matrices formatting is clear when displayed both in the file and on the screen.

File=MM_dynamic.cpp

Multiplication of 2 Matrices

Mathematical formula $$(A \times B)_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj}$$

Example $$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

Then

$$A \times B = \begin{bmatrix} (a_{11} \times b_{11} + a_{12} \times b_{21}) & (a_{11} \times b_{12} + a_{12} \times b_{22}) \\ (a_{21} \times b_{11} + a_{22} \times b_{21}) & (a_{21} \times b_{12} + a_{22} \times b_{22}) \end{bmatrix}$$

Solution of task 3.1

```cpp
#include <iostream>
#include <vector>
#include <fstream>
#include <cstdlib> // For rand() and srand()
#include <ctime>   // For time()

using namespace std;

int main() {
    srand(time(0)); // Seed for random number generation

    int rowsA, colsA, rowsB, colsB;
    cout << "Enter the number of rows for Matrix A: ";
    cin >> rowsA;
    cout << "Enter the number of columns for Matrix A: ";
    cin >> colsA;

    cout << "Enter the number of rows for Matrix B: ";
    cin >> rowsB;
    cout << "Enter the number of columns for Matrix B: ";
    cin >> colsB;

    if (colsA != rowsB) {
        cerr << "Matrix multiplication not possible. Number of columns of A must equal number of rows of B." << endl;
        return 1;
    }
```

```cpp
    // Generating matrices A and B
    vector<vector<int>> A(rowsA, vector<int>(colsA));
    vector<vector<int>> B(rowsB, vector<int>(colsB));

    cout << "Matrix A:\n";
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsA; ++j) {
            A[i][j] = rand() % 100 + 1;
            cout << A[i][j] << " ";
        }
        cout << endl;
    }

    cout << "\nMatrix B:\n";
    for (int i = 0; i < rowsB; ++i) {
        for (int j = 0; j < colsB; ++j) {
            B[i][j] = rand() % 100 + 1;
            cout << B[i][j] << " ";
        }
        cout << endl;
    }
```

```cpp
    // Multiplying matrices A and B
    vector<vector<int>> C(rowsA, vector<int>(colsB, 0));
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            for (int k = 0; k < colsA; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
```

Solution of task 3.1

```cpp
#include <iostream>
#include <vector>
#include <fstream>
#include <cstdlib> // For rand() and srand()
#include <ctime>   // For time()

using namespace std;

int main() {
    srand(time(0)); // Seed for random number gene

    int rowsA, colsA, rowsB, colsB;
    cout << "Enter the number of rows for Matrix A
    cin >> rowsA;
    cout << "Enter the number of columns for Matri
    cin >> colsA;

    cout << "Enter the number of rows for Matrix B
    cin >> rowsB;
    cout << "Enter the number of columns for Matri
    cin >> colsB;

    if (colsA != rowsB) {
        cerr << "Matrix multiplication not possibl
        return 1;
    }
```

```cpp
    // Generating matrices A and B
    vector<vector<int>> A(rowsA, vector<int>(colsA
    vector<vector<int>> B(rowsB, vector<int>(colsB

    cout << "Matrix A:\n";
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsA; ++j) {
            A[i][j] = rand() % 100 + 1;
            cout << A[i][j] << " ";
        }
        cout << endl;
    }

    cout << "\nMatrix B:\n";
    for (int i = 0; i < rowsB; ++i) {
        for (int j = 0; j < colsB; ++j) {
            B[i][j] = rand() % 100 + 1;
            cout << B[i][j] << " ";
        }
        cout << endl;
    }
```

```cpp
    // Multiplying matrices A and B
    vector<vector<int>> C(rowsA, vector<int>(colsB
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            for (int k = 0; k < colsA; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
```

```cpp
#include <iostream>
#include <vector>
#include <fstream>
#include <cstdlib> // For rand() and srand()
#include <ctime>   // For time()

using namespace std;

int main() {
    srand(time(0)); // Seed for random number generation

    int rowsA, colsA, rowsB, colsB;
    cout << "Enter the number of rows for Matrix A: ";
    cin >> rowsA;
    cout << "Enter the number of columns for Matrix A: ";
    cin >> colsA;

    cout << "Enter the number of rows for Matrix B: ";
    cin >> rowsB;
    cout << "Enter the number of columns for Matrix B: ";
    cin >> colsB;

    if (colsA != rowsB) {
        cerr << "Matrix multiplication not possible. Number of columns of A must equal number of rows of B." << endl;
        return 1;
    }
```

Solution of task 3.1

```cpp
#include <iostream>
#include <vector>
#include <fstream>
#include <cstdlib> // For rand() and srand()
#include <ctime>   // For time()

using namespace std;

int main() {
    srand(time(0)); // Seed for random number generation

    int rowsA, colsA, rowsB, colsB;
    cout << "Enter the number of rows for Matrix A: ";
    cin >> rowsA;
    cout << "Enter the number of columns for Matrix A: ";
    cin >> colsA;

    cout << "Enter the number of rows for Matrix B: ";
    cin >> rowsB;
    cout << "Enter the number of columns for Matrix B: ";
    cin >> colsB;

    if (colsA != rowsB) {
        cerr << "Matrix multiplication not possible. Number of columns of A must equal number of rows of B." << endl;
        return 1;
    }
```

```cpp
    // Generating matrices A and B
    vector<vector<int>> A(rowsA, vector<int>(colsA));
    vector<vector<int>> B(rowsB, vector<int>(colsB));

    cout << "Matrix A:\n";
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsA; ++j) {
            A[i][j] = rand() % 100 + 1;
            cout << A[i][j] << " ";
        }
        cout << endl;
    }

    cout << "\nMatrix B:\n";
    for (int i = 0; i < rowsB; ++i) {
        for (int j = 0; j < colsB; ++j) {
            B[i][j] = rand() % 100 + 1;
            cout << B[i][j] << " ";
        }
        cout << endl;
    }
```

```cpp
    // Multiplying matrices A and B
    vector<vector<int>> C(rowsA, vector<int>(colsB, 0));
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            for (int k = 0; k < colsA; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
```

Solution of task 3.1

```cpp
#include <iostream>
#include <vector>
#include <fstream>
#include <cstdlib> // For rand() and srand()
#include <ctime>   // For time()

using namespace std;

int main() {
    srand(time(0)); // Seed for random number generation

    int rowsA, colsA, rowsB, colsB;
    cout << "Enter the number of rows for Matrix A: ";
    cin >> rowsA;
    cout << "Enter the number of columns for Matrix A: ";
    cin >> colsA;

    cout << "Enter the number of rows for Matrix B: ";
    cin >> rowsB;
    cout << "Enter the number of columns for Matrix B: ";
    cin >> colsB;

    if (colsA != rowsB) {
        cerr << "Matrix multiplication not possible. Number of columns of A must equal number
        return 1;
    }
```

```cpp
    // Generating matrices A and B
    vector<vector<int>> A(rowsA, vector<int>(colsA));
    vector<vector<int>> B(rowsB, vector<int>(colsB));

    cout << "Matrix A:\n";
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsA; ++j) {
            A[i][j] = rand() % 100 + 1;
            cout << A[i][j] << " ";
        }
        cout << endl;
    }

    cout << "\nMatrix B:\n";
    for (int i = 0; i < rowsB; ++i) {
        for (int j = 0; j < colsB; ++j) {
            B[i][j] = rand() % 100 + 1;
            cout << B[i][j] << " ";
        }
        cout << endl;
    }
```

```cpp
    // Multiplying matrices A and B
    vector<vector<int>> C(rowsA, vector<int>(colsB, 0));
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            for (int k = 0; k < colsA; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
```

```cpp
// Generating matrices A and B
vector<vector<int>> A(rowsA, vector<int>(colsA));
vector<vector<int>> B(rowsB, vector<int>(colsB));

cout << "Matrix A:\n";
for (int i = 0; i < rowsA; ++i) {
    for (int j = 0; j < colsA; ++j) {
        A[i][j] = rand() % 100 + 1;
        cout << A[i][j] << " ";
    }
    cout << endl;
}

cout << "\nMatrix B:\n";
for (int i = 0; i < rowsB; ++i) {
    for (int j = 0; j < colsB; ++j) {
        B[i][j] = rand() % 100 + 1;
        cout << B[i][j] << " ";
    }
    cout << endl;
}
```

Solution of task 3.1

```cpp
#include <iostream>
#include <vector>
#include <fstream>
#include <cstdlib> // For rand() and srand()
#include <ctime>   // For time()

using namespace std;

int main() {
    srand(time(0)); // Seed for random number generation

    int rowsA, colsA, rowsB, colsB;
    cout << "Enter the number of rows for Matrix A: ";
    cin >> rowsA;
    cout << "Enter the number of columns for Matrix A: ";
    cin >> colsA;

    cout << "Enter the number of rows for Matrix B: ";
    cin >> rowsB;
    cout << "Enter the number of columns for Matrix B: ";
    cin >> colsB;

    if (colsA != rowsB) {
        cerr << "Matrix multiplication not possible. Number of columns of A must equal number of rows of B." << endl;
        return 1;
    }
```

```cpp
    // Generating matrices A and B
    vector<vector<int>> A(rowsA, vector<int>(colsA));
    vector<vector<int>> B(rowsB, vector<int>(colsB));

    cout << "Matrix A:\n";
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsA; ++j) {
            A[i][j] = rand() % 100 + 1;
            cout << A[i][j] << " ";
        }
        cout << endl;
    }

    cout << "\nMatrix B:\n";
    for (int i = 0; i < rowsB; ++i) {
        for (int j = 0; j < colsB; ++j) {
            B[i][j] = rand() % 100 + 1;
            cout << B[i][j] << " ";
        }
        cout << endl;
    }
```

```cpp
    // Multiplying matrices A and B
    vector<vector<int>> C(rowsA, vector<int>(colsB, 0));
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            for (int k = 0; k < colsA; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
```

Solution of task 3.1

```cpp
#include <iostream>
#include <vector>
#include <fstream>
#include <cstdlib> // For rand() and srand()
#include <ctime>   // For time()

using namespace std;

int main() {
    srand(time(0)); // Seed for random number generation

    int rowsA, colsA, rowsB, colsB;
    cout << "Enter the number of rows for Matrix A: ";
    cin >> rowsA;
    cout << "Enter the number of columns for Matrix A: ";
    cin >> colsA;

    cout << "Enter the number of rows for Matrix B: ";
    cin >> rowsB;
    cout << "Enter the number of columns for Matrix B: ";
    cin >> colsB;

    if (colsA != rowsB) {
        cerr << "Matrix multiplication not possible. Number of colum
        return 1;
    }
```

```cpp
    // Generating matrices A and B
    vector<vector<int>> A(rowsA, vector<int>(colsA));
    vector<vector<int>> B(rowsB, vector<int>(colsB));

    cout << "Matrix A:\n";
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsA; ++j) {
            A[i][j] = rand() % 100 + 1;
            cout << A[i][j] << " ";
        }
        cout << endl;
    }

    cout << "\nMatrix B:\n";
    for (int i = 0; i < rowsB; ++i) {
        for (int j = 0; j < colsB; ++j) {
            B[i][j] = rand() % 100 + 1;
            cout << B[i][j] << " ";
        }
        cout << endl;
    }
```

```cpp
    // Multiplying matrices A and B
    vector<vector<int>> C(rowsA, vector<int>(colsB, 0));
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            for (int k = 0; k < colsA; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
```

```cpp
// Multiplying matrices A and B
vector<vector<int>> C(rowsA, vector<int>(colsB, 0));
for (int i = 0; i < rowsA; ++i) {
    for (int j = 0; j < colsB; ++j) {
        for (int k = 0; k < colsA; ++k) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Solution of task 3.1

```
    // Displaying the result matrix
    cout << "\nResult of A * B:\n";
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            cout << C[i][j] << " ";
        }
        cout << endl;
    }
```

```
   // Writing matrices A, B, and C to a file
    ofstream outFile("matrix_multiplication.txt");
    if (outFile.is_open()) {
        outFile << "Matrix A:\n";
        for (int i = 0; i < rowsA; ++i) {
            for (int j = 0; j < colsA; ++j) {
                outFile << A[i][j] << " ";
            }
            outFile << "\n";
        }

        outFile << "\nMatrix B:\n";
        for (int i = 0; i < rowsB; ++i) {
            for (int j = 0; j < colsB; ++j) {
                outFile << B[i][j] << " ";
            }
            outFile << "\n";
        }

        outFile << "\nResult of A * B:\n";
        for (int i = 0; i < rowsA; ++i) {
            for (int j = 0; j < colsB; ++j) {
                outFile << C[i][j] << " ";
            }
            outFile << "\n";
        }
        outFile.close();
    } else {
        cerr << "Unable to open file for writing." << endl;
        return 1;
    }

    return 0;
}
```

Solution of task 3.1

```
    // Displaying the result matrix
    cout << "\nResult of A * B:\n";
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            cout << C[i][j] << " ";
        }
        cout << endl;
    }
```

```
   // Writing matrices A, B, and C to a fi
    ofstream outFile("matrix_multiplicatio
    if (outFile.is_open()) {
        outFile << "Matrix A:\n";
        for (int i = 0; i < rowsA; ++i) {
            for (int j = 0; j < colsA; ++j
                outFile << A[i][j] << " ";
            }
            outFile << "\n";
        }

        outFile << "\nMatrix B:\n";
        for (int i = 0; i < rowsB; ++i) {
            for (int j = 0; j < colsB; ++j
                outFile << B[i][j] << " ";
            }
            outFile << "\n";
        }

        outFile << "\nResult of A * B:\n";
        for (int i = 0; i < rowsA; ++i) {
            for (int j = 0; j < colsB; ++j) {
                outFile << C[i][j] << " ";
            }
            outFile << "\n";
        }
        outFile.close();
    } else {
        cerr << "Unable to open file for writing." << endl;
        return 1;
    }

    return 0;
}
```

```
    // Displaying the result matrix
    cout << "\nResult of A * B:\n";
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            cout << C[i][j] << " ";
        }
        cout << endl;
    }
```

Solution of task 3.1

```
    // Displaying the result matrix
    cout << "\nResult of A * B:\n";
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            cout << C[i][j] << " ";
        }
        cout << endl;
    }
```

```
   // Writing matrices A, B, and C to a file
    ofstream outFile("matrix_multiplication.txt");
    if (outFile.is_open()) {
        outFile << "Matrix A:\n";
        for (int i = 0; i < rowsA; ++i) {
            for (int j = 0; j < colsA; ++j) {
                outFile << A[i][j] << " ";
            }
            outFile << "\n";
        }

        outFile << "\nMatrix B:\n";
        for (int i = 0; i < rowsB; ++i) {
            for (int j = 0; j < colsB; ++j) {
                outFile << B[i][j] << " ";
            }
            outFile << "\n";
        }

        outFile << "\nResult of A * B:\n";
        for (int i = 0; i < rowsA; ++i) {
            for (int j = 0; j < colsB; ++j) {
                outFile << C[i][j] << " ";
            }
            outFile << "\n";
        }
        outFile.close();
    } else {
        cerr << "Unable to open file for writing." << endl;
        return 1;
    }

    return 0;
}
```

Solution of task 3.1

```
    // Displaying the result matrix
    cout << "\nResult of A * B:\n";
    for (int i = 0; i < rowsA; ++i) {
        for (int j = 0; j < colsB; ++j) {
            cout << C[i][j] << " ";
        }
        cout << endl;
    }
```

```
   // Writing matrices A, B, and C to a file
    ofstream outFile("matrix_multiplication.txt");
    if (outFile.is_open()) {
        outFile << "Matrix A:\n";
        for (int i = 0; i < rowsA; ++i) {
            for (int j = 0; j < colsA; ++j) {
                outFile << A[i][j] << " ";
            }
            outFile << "\n";
        }

        outFile << "\nMatrix B:\n";
        for (int i = 0; i < rowsB; ++i) {
            for (int j = 0; j < colsB; ++j) {
                outFile << B[i][j] << " ";
            }
            outFile << "\n";
        }

        outFile << "\nResult of A * B:\n";
        for (int i = 0; i < rowsA; ++i) {
            for (int j = 0; j < colsB; ++j) {
                outFile << C[i][j] << " ";
            }
            outFile << "\n";
        }
        outFile.close();
    } else {
        cerr << "Unable to open file for writing."
        return 1;
    }

    return 0;
}
```

```
   // Writing matrices A, B, and C to a file
    ofstream outFile("matrix_multiplication.txt");
    if (outFile.is_open()) {
        outFile << "Matrix A:\n";
        for (int i = 0; i < rowsA; ++i) {
            for (int j = 0; j < colsA; ++j) {
                outFile << A[i][j] << " ";
            }
            outFile << "\n";
        }

        outFile << "\nMatrix B:\n";
        for (int i = 0; i < rowsB; ++i) {
            for (int j = 0; j < colsB; ++j) {
                outFile << B[i][j] << " ";
            }
            outFile << "\n";
        }

        outFile << "\nResult of A * B:\n";
        for (int i = 0; i < rowsA; ++i) {
            for (int j = 0; j < colsB; ++j) {
                outFile << C[i][j] << " ";
            }
            outFile << "\n";
        }
        outFile.close();
    } else {
        cerr << "Unable to open file for writing." << endl;
        return 1;
    }

    return 0;
}
```

Solution of task 3.1

Outputs

```
Enter the number of rows for Matrix A: 3
Enter the number of columns for Matrix A: 3
Enter the number of rows for Matrix B: 3
Enter the number of columns for Matrix B: 3
Matrix A:
69 98 4
100 20 61
85 13 64

Matrix B:
90 19 54
30 37 57
7 55 68

Result of A * B:
9178 5157 9584
10027 5995 10688
8488 5616 9683
```

```
Enter the number of rows for Matrix A: 3
Enter the number of columns for Matrix A: 4
Enter the number of rows for Matrix B: 4
Enter the number of columns for Matrix B: 3
Matrix A:
31 56 72 18
55 55 48 46
14 70 37 46

Matrix B:
17 15 58
60 69 35
77 100 35
85 25 12

Result of A * B:
10961 11979 6494
11841 10570 7347
11197 9890 5109
```

```
Enter the number of rows for Matrix A: 3
Enter the number of columns for Matrix A: 3
Enter the number of rows for Matrix B: 4
Enter the number of columns for Matrix B: 4
Matrix multiplication not possible. Number of columns of A must equal number of rows of B.
```

Overview Semester B

NUMERICAL METHODS AND ALGORITHMS

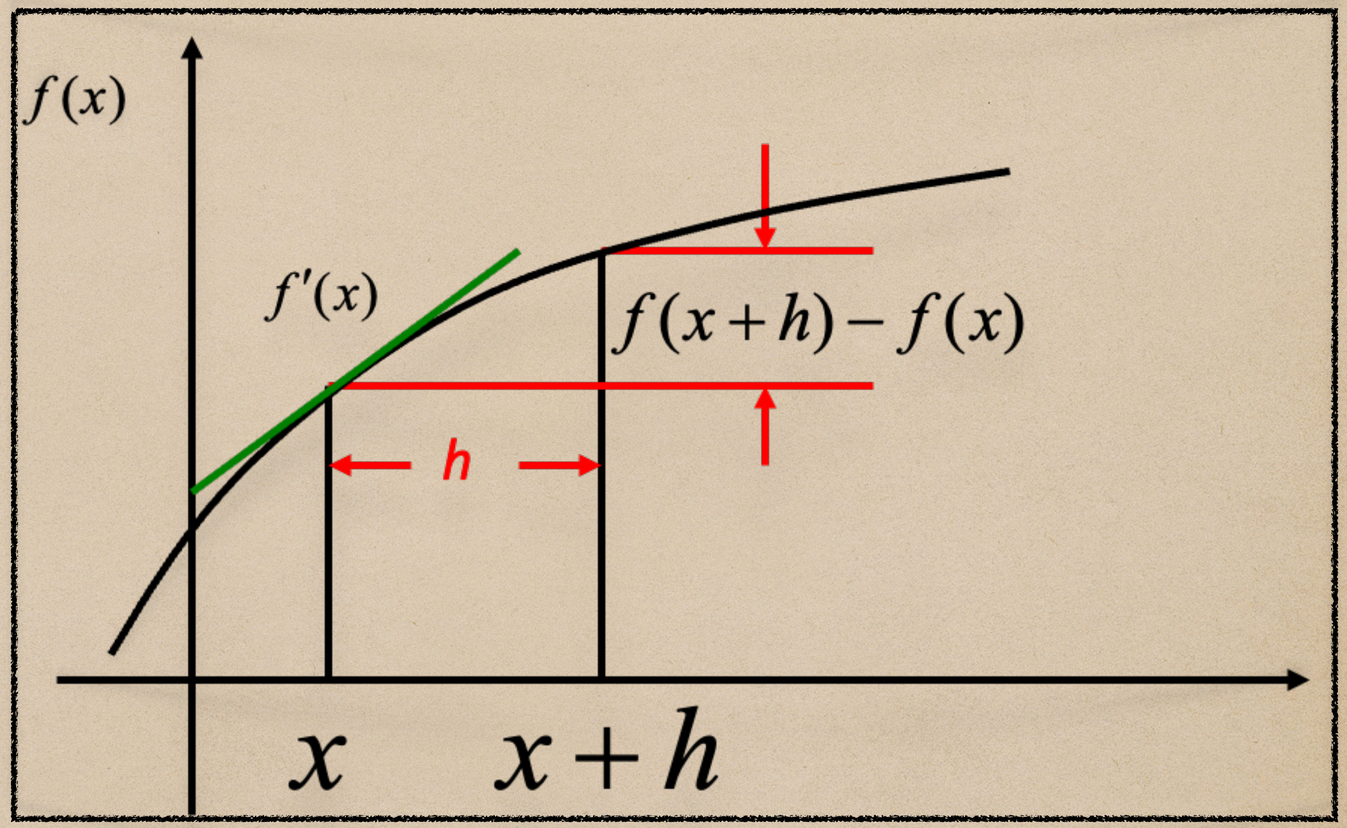
- Random Numbers
- Sorting algorithm
- Numerical Differentiation
- Root of a equations
  - Bisection Method
  - Newton Method
  - Secant Method
- Numerical Integration
- Numerical integration of Ordinary Differential Equation (first order)
- Linear Algebra
  - Product of scalar with vectors/matrix
  - Product of two matrix

C++ Coding

- Open, read and write a file in C++
- Dynamical arrays

Numerical differentiation

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$



Numerical differentiation

Forward difference formula

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

- h is a small step size
- (f(x + h) - f(x) represents the change in the function over the interval h)

Numerical differentiation

Forward difference formula

The Taylor expansion of f(x+h) around x is

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + O(h^4)$$

then

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2}f''(x) - \frac{h^2}{6}f'''(x) + O(h^3)$$

by truncating after the first term:

$$\boxed{f'(x) \approx \frac{f(x+h) - f(x)}{h}}$$

Numerical differentiation

Forward difference formula

The leading error term is

$$Error = -\frac{h}{2} f''(x) + O(h^2)$$

The truncation error is of the order of $O(h)$, making this a first-order method

Numerical differentiation

Backward difference formula

$$\boxed{f'(x) \approx \frac{f(x) - f(x-h)}{h}}$$

The Taylor expansion of f(x-h) around x is:

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + O(h^4)$$

then

$$f'(x) = \frac{f(x) - f(x-h)}{h} - \frac{h}{2}f''(x) + \frac{h^2}{6}f'''(x) + O(h^3)$$

Numerical differentiation

Backward difference formula

Considering only the first term:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

The leading error term is

$$Error = -\frac{h}{2} f''(x) + O(h^2)$$

The truncation error is of the order of $O(h)$, making this a first-order method.

Numerical differentiation

Central difference formula

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

The Taylor expansion of f(x+h) around x is

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + O(h^4)$$

The Taylor expansion of f(x-h) around x is:

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + O(h^4)$$

Numerical differentiation

Central difference formula

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + O(h^4)$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + O(h^4)$$

Substrating the second equation from the first:

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{2h^3}{6}f'''(x) + O(h^4)$$

then

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6}f'''(x) + O(h^4)$$

Numerical differentiation

Central difference formula

The error is

$$\text{Error} = -\frac{h^2}{6} f'''(x) + O(h^4)$$

The truncation error is of the order of $O(h^2)$, making this a second-order method.

Numerical differentiation

Summary

Forward difference formula

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Backward difference formula

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

Central difference formula

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Numerical differentiation

for 3 points $(x_0, x_0 + h, x_0 + 2h)$

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(x_0) + \frac{h^3}{6} f'''(\xi_1)$$

$$f(x_0 + 2h) = f(x_0) + 2hf'(x_0) + 2h^2 f''(x_0) + \frac{4h^3}{3} f'''(\xi_2)$$

$$Af(x_0) + Bf(x_0 + h) + Cf(x_0 + 2h)$$

$$= Af(x_0) + B\left(f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(x_0)\right) + C\left(f(x_0) + 2hf'(x_0) + 2h^2 f''(x_0)\right)$$

Numerical differentiation

Central difference formula

$$= (A + B + C)f(x_0) + (B + 2C)hf'(x_0) + \left(\frac{B}{2} + 2C\right) h^2 f''(x_0) + O(h^3)$$

$$\begin{cases} A + B + C = 0 & \text{(eliminate the constant term } f(x_0)) \\ B + 2C = 1 & \text{(ensure the coefficient of } f'(x_0) \text{ is correct)} \\ \frac{B}{2} + 2C = 0 & \text{(eliminate the second derivative term } f''(x_0)) \end{cases}$$

$$A = -\frac{3}{2}, \quad B = 2, \quad C = -\frac{1}{2}$$

$$f'(x_0) \approx \frac{-3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h)}{2h}$$

Exercise: Differentiate log (x)

1. User inputs
   - The value of x_0 (the point where the derivative is approssimate)
   - The starting value of h
   - The number of iterations

example1.cpp

2. Output
   - Display iteration number, h, approximate derivative, exact, derivative, and error on the screen.
   - Write only the approximation and iteration number to a file (exercise_results.txt).

Solution:

```cpp
#include <iostream>
#include <fstream>
#include <cmath>    // For log() function
#include <iomanip>  // For setting precision

// Function to approximate ln(x) using the forward difference method
double forward_difference_ln(double x0, double h) {
    return (log(x0 + h) - log(x0)) / h;
}

int main() {
    double x0, h_start, dh;
    int num_iterations;

    // User input for x0, starting h, and number of iterations
    std::cout << "Enter the value of x0: ";
    std::cin >> x0;
    std::cout << "Enter the starting value of h: ";
    std::cin >> h_start;
    std::cout << "Enter the number of iterations: ";
    std::cin >> num_iterations;

    dh = h_start / num_iterations;

    // Open file to write results
    std::ofstream outfile("exercise_results.txt");
    if (!outfile.is_open()) {
        std::cerr << "Error: Unable to open file for writing." << std::endl;
        return 1;
    }
```

Solution:

```
    std::cout << std::fixed << std::setprecision(6);
    outfile << std::fixed << std::setprecision(6);

    // Loop for the specified number of iterations
    double h = h_start;
    for (int i = 1; i <= num_iterations; i++) {
        double approx_derivative = forward_difference_ln(x0, h);
        double exact_derivative = 1 / x0;
        double error = fabs(approx_derivative - exact_derivative);

        // Display the results on the screen
        std::cout << "Iteration: " << i << ", h: " << h
                  << ", Approximate: " << approx_derivative
                  << ", Exact: " << exact_derivative
                  << ", Error: " << error << std::endl;

        // Write only the approximation and iteration number to the file
        outfile << approx_derivative << " " << i << "\n";

        h -= dh; // Decrease h by dh for the next iteration
    }

    outfile.close();
    std::cout << "Results have been written to exercise_results.txt" << std::endl;

    return 0;
}
```

Output

```
Iteration: 1, h: 1.000000, Approximate: 0.441833, Exact: 0.555556, Error: 0.113723
Iteration: 2, h: 0.990000, Approximate: 0.442682, Exact: 0.555556, Error: 0.112874
Iteration: 3, h: 0.980000, Approximate: 0.443535, Exact: 0.555556, Error: 0.112021
Iteration: 4, h: 0.970000, Approximate: 0.444392, Exact: 0.555556, Error: 0.111163
Iteration: 5, h: 0.960000, Approximate: 0.445254, Exact: 0.555556, Error: 0.110301
Iteration: 6, h: 0.950000, Approximate: 0.446120, Exact: 0.555556, Error: 0.109435
Iteration: 7, h: 0.940000, Approximate: 0.446991, Exact: 0.555556, Error: 0.108565
Iteration: 8, h: 0.930000, Approximate: 0.447866, Exact: 0.555556, Error: 0.107690
Iteration: 9, h: 0.920000, Approximate: 0.448745, Exact: 0.555556, Error: 0.106811
Iteration: 10, h: 0.910000, Approximate: 0.449629, Exact: 0.555556, Error: 0.105927
Iteration: 11, h: 0.900000, Approximate: 0.450517, Exact: 0.555556, Error: 0.105039
Iteration: 12, h: 0.890000, Approximate: 0.451410, Exact: 0.555556, Error: 0.104146
Iteration: 13, h: 0.880000, Approximate: 0.452307, Exact: 0.555556, Error: 0.103249
Iteration: 14, h: 0.870000, Approximate: 0.453209, Exact: 0.555556, Error: 0.102347
Iteration: 15, h: 0.860000, Approximate: 0.454116, Exact: 0.555556, Error: 0.101440
Iteration: 16, h: 0.850000, Approximate: 0.455027, Exact: 0.555556, Error: 0.100529
Iteration: 17, h: 0.840000, Approximate: 0.455943, Exact: 0.555556, Error: 0.099612
Iteration: 18, h: 0.830000, Approximate: 0.456864, Exact: 0.555556, Error: 0.098691
Iteration: 19, h: 0.820000, Approximate: 0.457790, Exact: 0.555556, Error: 0.097766
Iteration: 20, h: 0.810000, Approximate: 0.458720, Exact: 0.555556, Error: 0.096835
Iteration: 21, h: 0.800000, Approximate: 0.459656, Exact: 0.555556, Error: 0.095900
Iteration: 22, h: 0.790000, Approximate: 0.460596, Exact: 0.555556, Error: 0.094959
Iteration: 23, h: 0.780000, Approximate: 0.461542, Exact: 0.555556, Error: 0.094014
Iteration: 24, h: 0.770000, Approximate: 0.462493, Exact: 0.555556, Error: 0.093063
Iteration: 25, h: 0.760000, Approximate: 0.463448, Exact: 0.555556, Error: 0.092107
Iteration: 26, h: 0.750000, Approximate: 0.464409, Exact: 0.555556, Error: 0.091147
Iteration: 27, h: 0.740000, Approximate: 0.465375, Exact: 0.555556, Error: 0.090181
Iteration: 28, h: 0.730000, Approximate: 0.466346, Exact: 0.555556, Error: 0.089209
Iteration: 29, h: 0.720000, Approximate: 0.467323, Exact: 0.555556, Error: 0.088233
Iteration: 30, h: 0.710000, Approximate: 0.468304, Exact: 0.555556, Error: 0.087251
Iteration: 31, h: 0.700000, Approximate: 0.469292, Exact: 0.555556, Error: 0.086264
Iteration: 32, h: 0.690000, Approximate: 0.470284, Exact: 0.555556, Error: 0.085271
Iteration: 33, h: 0.680000, Approximate: 0.471282, Exact: 0.555556, Error: 0.084273
Iteration: 34, h: 0.670000, Approximate: 0.472286, Exact: 0.555556, Error: 0.083270
Iteration: 35, h: 0.660000, Approximate: 0.473295, Exact: 0.555556, Error: 0.082261
Iteration: 36, h: 0.650000, Approximate: 0.474310, Exact: 0.555556, Error: 0.081246
Iteration: 37, h: 0.640000, Approximate: 0.475330, Exact: 0.555556, Error: 0.080225
Iteration: 38, h: 0.630000, Approximate: 0.476356, Exact: 0.555556, Error: 0.079199
Iteration: 39, h: 0.620000, Approximate: 0.477389, Exact: 0.555556, Error: 0.078167
Iteration: 40, h: 0.610000, Approximate: 0.478426, Exact: 0.555556, Error: 0.077129
Iteration: 41, h: 0.600000, Approximate: 0.479470, Exact: 0.555556, Error: 0.076085
Iteration: 42, h: 0.590000, Approximate: 0.480520, Exact: 0.555556, Error: 0.075036
Iteration: 43, h: 0.580000, Approximate: 0.481576, Exact: 0.555556, Error: 0.073980
Iteration: 44, h: 0.570000, Approximate: 0.482637, Exact: 0.555556, Error: 0.072918
Iteration: 45, h: 0.560000, Approximate: 0.483705, Exact: 0.555556, Error: 0.071850
```

Task 4.1: Differentiate log (x)

1.User inputs

- The value of x_0 =1.8 (the point where the derivative is approssimate)
- The starting value of h
- The number of iterations

task4_1.cpp

2.Output

- Display each iteration with approximation from all three methods
- Their corresponding errors.
- Write interation numbers and approximation to a file (difference_exercise_results.txt).

Task 4.1 Differentiate log (x)

```
Enter the value of x0: 1.8
Enter the starting value of h: 1
Enter the number of iterations: 500
Iteration: 1, h: 1.000000
  Forward Approximate: 0.441833, Error: 0.113723
  Backward Approximate: 0.810930, Error: 0.255375
  Central Approximate: 0.626381, Error: 0.070826
Iteration: 2, h: 0.998000
  Forward Approximate: 0.442002, Error: 0.113553
  Backward Approximate: 0.810053, Error: 0.254498
  Central Approximate: 0.626028, Error: 0.070472
Iteration: 3, h: 0.996000
  Forward Approximate: 0.442172, Error: 0.113384
  Backward Approximate: 0.809179, Error: 0.253624
  Central Approximate: 0.625676, Error: 0.070120
Iteration: 4, h: 0.994000
  Forward Approximate: 0.442342, Error: 0.113214
  Backward Approximate: 0.808308, Error: 0.252752
  Central Approximate: 0.625325, Error: 0.069769
Iteration: 5, h: 0.992000
  Forward Approximate: 0.442512, Error: 0.113044
  Backward Approximate: 0.807439, Error: 0.251884
  Central Approximate: 0.624976, Error: 0.069420
Iteration: 6, h: 0.990000
  Forward Approximate: 0.442682, Error: 0.112874
  Backward Approximate: 0.806573, Error: 0.251018
  Central Approximate: 0.624628, Error: 0.069072
Iteration: 7, h: 0.988000
  Forward Approximate: 0.442852, Error: 0.112704
  Backward Approximate: 0.805710, Error: 0.250155
  Central Approximate: 0.624281, Error: 0.068726
Iteration: 8, h: 0.986000
  Forward Approximate: 0.443023, Error: 0.112533
  Backward Approximate: 0.804849, Error: 0.249294
  Central Approximate: 0.623936, Error: 0.068380
```

Task 4.2: Differentiate sin (x)

1.User inputs

- The value of x_0 (the point where the derivative is approximate)
- The starting value of h
- The number of iterations N=500

task4_2.cpp

2.Output

- Display each iteration with approximation from all three methods
- Their corresponding errors, value of h.
- Absolute and relative error
- Write interation numbers and approximation to a file (difference_exercise_results.txt).

Task 4.2: Differentiate sin (x)

Extra tasks-

- Identify which method provides the smallest error
- Determine how the error decreases as h becomes smaller

Modify h and N.

- h=0.5 and N=500
- h=0.1 and N=2000
- Observe how the results change.

Graphical representation

- Import the data from the file to Excel or using Matlab,Python.
- Plot error vs iteration number for the three different methods.

Questions?