# Gaussian Elimination Solver

1.0

Generated by Doxygen 1.12.0

# Chapter 1

# Gaussian Elimination Solver

This project solves systems of linear equations using Gaussian elimination.

The program reads matrices from `.in` files, performs Gaussian elimination with partial pivoting, determines the rank and consistency of the system, and displays the solution. It allows multiple runs and interacts with the user for input and exit control.

### 1.0.1 Features

- Gaussian elimination with partial pivoting
- Rank determination and consistency check
- Handles cases with no solution, unique solution, or infinitely many solutions

### 1.0.2 Usage

1. Provide a matrix in an `.in` file.
2. The program reads the matrix and applies Gaussian elimination.
3. The user can run the program multiple times.

# Chapter 2

# File Index

## 2.1 File List

Here is a list of all files with brief descriptions:

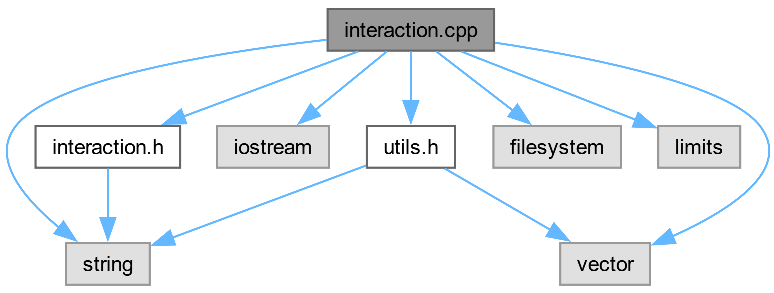
# Chapter 3

# File Documentation

## 3.1 interaction.cpp File Reference

Implementation of user interaction functions.

```
#include "interaction.h"
#include <iostream>
#include <vector>
#include <string>
#include <filesystem>
#include <limits>
#include "utils.h"
```
Include dependency graph for interaction.cpp:



### Functions

- string SelectInputFile ()

  *Allows the user to select an input .in file from the current directory.*
- char AskRunAgain ()

  *Asks the user if they want to run the program again.*
- void WaitForExit ()

  *Waits for the user to press Enter before exiting.*

### 3.1.1 Detailed Description

Implementation of user interaction functions.

**Author**

Gilbert Young

**Date**

2024/09/25

This file implements the functions responsible for interacting with the user, including selecting input files, prompting whether to run the program again, and waiting for the user to exit. These functions guide the flow of the program based on user input.

### 3.1.2 Function Documentation

#### 3.1.2.1 AskRunAgain()

```
char AskRunAgain ()
```

Asks the user if they want to run the program again.

**Returns**

char The user's choice ('y', 'Y', 'n', 'N').

```
{
    char choice;
    while (true)
    {
        cout « "\nDo you want to run the program again? (y/n): ";
        cin » choice;

        if (choice == 'y' || choice == 'Y' || choice == 'n' || choice == 'N')
        {
            break;
        }
        else
        {
            cout « "Invalid input. Please enter 'y' or 'n'." « endl;
        }
    }
    return choice;
}
```

#### 3.1.2.2 SelectInputFile()

```
string SelectInputFile ()
```

Allows the user to select an input .in file from the current directory.

**Returns**

std::string The name of the selected file. Empty string if no file is selected.

```
{
    vector<string> in_files;
    for (const auto &entry : filesystem::directory_iterator(filesystem::current_path()))
    {
        if (entry.is_regular_file())
        {
            string filename = entry.path().filename().string();
            if (filename.size() >= 3 && filename.substr(filename.size() - 3) == ".in")
            {
                in_files.push_back(filename);
            }
        }
    }

    string selected_file;
    if (in_files.empty())
    {
        cout « "No .in files found in the current directory." « endl;
        return "";
    }
    else if (in_files.size() == 1)
    {
        selected_file = in_files[0];
        cout « "Found one .in file: " « selected_file « " . Automatically selecting it." « endl;
    }
    else
    {
        cout « "Multiple .in files found. Please select one:" « endl;
        for (size_t i = 0; i < in_files.size(); i++)
        {
            cout « i + 1 « ". " « in_files[i] « endl;
        }
        int file_choice;
        // Improved input validation
        while (true)
        {
            cout « "Enter the number of the file you want to use (1-" « in_files.size() « "): ";
            cin » file_choice;

            if (cin.fail() || file_choice < 1 || file_choice > static_cast<int>(in_files.size()))
            {
                cin.clear();                                         // Clear error flags
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
                cout « "Invalid input. Please enter a number between 1 and " « in_files.size() « "." « endl;
            }
            else
            {
                break;
            }
        }
        selected_file = in_files[file_choice - 1];
    }
    cout « endl;
    return selected_file;
}
```

### 3.1.2.3 WaitForExit()

```
void WaitForExit ()
```

Waits for the user to press Enter before exiting.

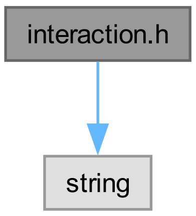
```
{
    cout « "\nPress Enter to exit...";
    cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
    cin.get();                                           // Wait for Enter key
}
```

## 3.2 interaction.h File Reference

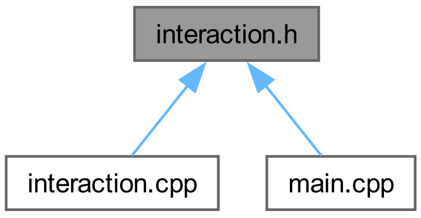
User interaction functions.

`#include <string>`

Include dependency graph for interaction.h:



This graph shows which files directly or indirectly include this file:



**Functions**

- std::string SelectInputFile ()

  *Allows the user to select an input .in file from the current directory.*
- char AskRunAgain ()

  *Asks the user if they want to run the program again.*
- void WaitForExit ()

  *Waits for the user to press Enter before exiting.*

### 3.2.1 Detailed Description

User interaction functions.

**Author**

Gilbert Young

**Date**

2024/09/25

## 3.2.2 Function Documentation

### 3.2.2.1 AskRunAgain()

```
char AskRunAgain ()
```

Asks the user if they want to run the program again.

#### Returns

char The user's choice ('y', 'Y', 'n', 'N').task

char The user's choice ('y', 'Y', 'n', 'N').

```
{
    char choice;
    while (true)
    {
        cout « "\nDo you want to run the program again? (y/n): ";
        cin » choice;

        if (choice == ’y’ || choice == ’Y’ || choice == ’n’ || choice == ’N’)
        {
            break;
        }
        else
        {
            cout « "Invalid input. Please enter ’y’ or ’n’." « endl;
        }
    }
    return choice;
}
```

### 3.2.2.2 SelectInputFile()

```
std::string SelectInputFile ()
```

Allows the user to select an input .in file from the current directory.

#### Returns

std::string The name of the selected file. Empty string if no file is selected.

```
{
    vector<string> in_files;
    for (const auto &entry : filesystem::directory_iterator(filesystem::current_path()))
    {
        if (entry.is_regular_file())
        {
            string filename = entry.path().filename().string();
            if (filename.size() >= 3 && filename.substr(filename.size() - 3) == ".in")
            {
                in_files.push_back(filename);
            }
        }
    }

    string selected_file;
    if (in_files.empty())
    {
        cout « "No .in files found in the current directory." « endl;
        return "";
    }
    else if (in_files.size() == 1)
    {
        selected_file = in_files[0];
        cout « "Found one .in file: " « selected_file « " . Automatically selecting it." « endl;
    }
    else
    {
        cout « "Multiple .in files found. Please select one:" « endl;
        for (size_t i = 0; i < in_files.size(); i++)
        {
            cout « i + 1 « ". " « in_files[i] « endl;
        }
        int file_choice;
        // Improved input validation
        while (true)
        {
            cout « "Enter the number of the file you want to use (1-" « in_files.size() « "): ";
            cin » file_choice;

            if (cin.fail() || file_choice < 1 || file_choice > static_cast<int>(in_files.size()))
            {
                cin.clear();                                         // Clear error flags
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
                cout « "Invalid input. Please enter a number between 1 and " « in_files.size() « "." « endl;
            }
            else
            {
                break;
            }
        }
        selected_file = in_files[file_choice - 1];
    }
    cout « endl;
    return selected_file;
}
```

#### 3.2.2.3 WaitForExit()

```
void WaitForExit ()
```

Waits for the user to press Enter before exiting.

```
{
    cout « "\nPress Enter to exit...";
    cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
    cin.get();                                           // Wait for Enter key
}
```

## 3.3 interaction.h

Go to the documentation of this file.

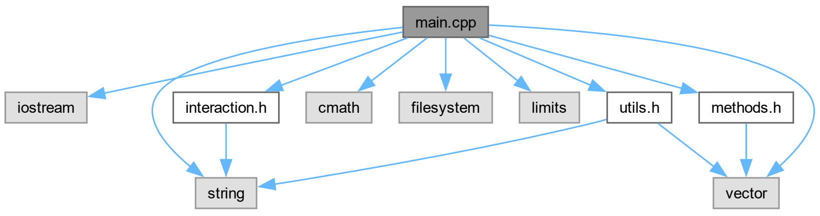
```

#ifndef INTERACTION_H
#define INTERACTION_H

#include <string>

std::string SelectInputFile();

char AskRunAgain();

void WaitForExit();

#endif // INTERACTION_H
```

## 3.4 main.cpp File Reference

Entry point for the Gaussian Elimination Solver project.

```
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include <filesystem>
#include <limits>
#include "utils.h"
```

```
#include "methods.h"
#include "interaction.h"
```
Include dependency graph for main.cpp:



## Functions

- int main ()

## 3.4.1 Detailed Description

Entry point for the Gaussian Elimination Solver project.

**Author**

Gilbert Young

**Date**

2024/09/25

## 3.4.2 Function Documentation

### 3.4.2.1 main()

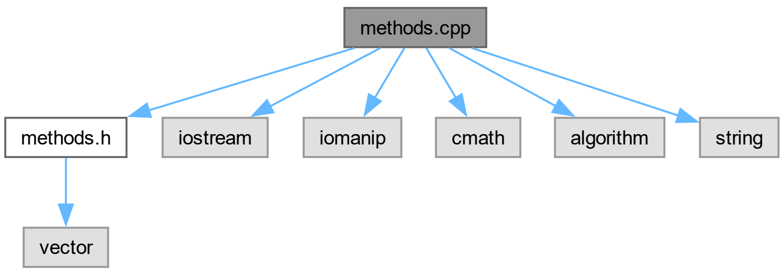
```
int main ()
{
    char choice;
    do
    {
        string selected_file = SelectInputFile();
        if (selected_file.empty())
        {
            return 1; // File selection failed
        }

        vector<vector<double» matrix;
        int rows, cols;
        if (!InitMatrix(matrix, selected_file, rows, cols))
        {
            return 1; // Matrix initialization failed
        }

        ShowEquations(matrix, rows, cols);
        cout « "Starting Gaussian elimination process..." « endl;
        int exchange_count = GaussianElimination(matrix, rows, cols);
        cout « "Gaussian elimination completed." « endl
             « endl;

        int rank = DetermineRank(matrix, rows, cols);
        bool consistent = CheckConsistency(matrix, rows, cols);

        if (!consistent)
        {
            cout « "The system of equations is inconsistent and has no solution." « endl;
        }
        else if (rank < (cols - 1))
        {
            ShowGeneralSolution(matrix, rows, cols, rank);
        }
        else
        {
            vector<double> solution;
            bool solvable = BackSubstitution(matrix, rows, cols, solution);
            if (solvable)
            {
                DisplaySolution(solution);
            }
            else
            {
                cout « "The system of equations is inconsistent and has no solution." « endl;
            }
        }

        choice = AskRunAgain();

    } while (choice == 'y' || choice == 'Y');

    WaitForExit();
    return 0;
}
```

## 3.5 methods.cpp File Reference

Implementation of computational functions for solving linear systems.

```
#include "methods.h"
#include <iostream>
#include <iomanip>
#include <cmath>
#include <algorithm>
#include <string>
```

Include dependency graph for methods.cpp:



### Functions

- int Pivoting (const vector< vector< double > > &m, int current_row, int total_rows)
- void Exchange (vector< vector< double > > &m, int row1, int row2)

- bool Eliminate (vector< vector< double > > &m, int current_row, int total_rows, int total_cols)
- int GaussianElimination (vector< vector< double > > &m, int rows, int cols)
- bool BackSubstitution (const vector< vector< double > > &m, int rows, int cols, vector< double > &solution)
- int DetermineRank (const vector< vector< double > > &m, int rows, int cols)
- vector< int > IdentifyPivots (const vector< vector< double > > &m, int rows, int cols)
- void ShowGeneralSolution (const vector< vector< double > > &m, int rows, int cols, int rank)

### 3.5.1 Detailed Description

Implementation of computational functions for solving linear systems.

**Author**

Gilbert Young

**Date**

2024/09/25

This file implements key algorithms such as Gaussian elimination with partial pivoting, back-substitution, and rank determination. It also includes functionality to display the general solution when the system has infinitely many solutions.

### 3.5.2 Function Documentation

#### 3.5.2.1 BackSubstitution()

```
bool BackSubstitution (
            const vector< vector< double > > & m,
            int rows,
            int cols,
            vector< double > & solution)
{
    solution.assign(cols - 1, 0.0);
    cout « "Starting back-substitution process..." « endl;
    for (int i = rows - 1; i >= 0; i--)
    {
        // Find the first non-zero coefficient in the row
        int pivot_col = -1;
        for (int j = 0; j < cols - 1; j++)
        {
            if (fabs(m[i][j]) > 1e-12)
            {
                pivot_col = j;
                break;
            }
        }

        if (pivot_col == -1)
        {
            if (fabs(m[i][cols - 1]) > 1e-12)
            {
                // Inconsistent equation
                return false;
            }
            else
            {
                // 0 = 0, skip
                continue;
            }
        }

        double rhs = m[i][cols - 1];
        cout « "Calculating x" « pivot_col + 1 « ":" « endl;
        for (int j = pivot_col + 1; j < cols - 1; j++)
        {
            cout « "    " « fixed « setprecision(4) « m[i][j] « " * x" « j + 1
                 « " = " « m[i][j] * solution[j] « endl;
            rhs -= m[i][j] * solution[j];
        }
        cout « "    RHS after subtraction = " « rhs « endl;
        solution[pivot_col] = rhs / m[i][pivot_col];
        cout « "    x" « pivot_col + 1 « " = " « rhs « " / " « m[i][pivot_col]
             « " = " « fixed « setprecision(4) « solution[pivot_col] « endl
             « endl;
    }
    return true;
}
```

### 3.5.2.2 DetermineRank()

```
int DetermineRank (
               const vector< vector< double > > & m,
               int rows,
               int cols)
{
    int rank = 0;
    for (int i = 0; i < rows; i++)
    {
        bool non_zero = false;
        for (int j = 0; j < cols - 1; j++)
        {
            if (fabs(m[i][j]) > 1e-12)
            {
                non_zero = true;
                break;
            }
        }
        if (non_zero)
            rank++;
    }
    return rank;
}
```

### 3.5.2.3 Eliminate()

```
bool Eliminate (
               vector< vector< double > > & m,
               int current_row,
               int total_rows,
               int total_cols)
{
    for (int i = current_row + 1; i < total_rows; i++)
    {
        if (fabs(m[current_row][current_row]) < 1e-12)
        {
            // Pivot is too small, cannot eliminate
            return false;
        }
        double factor = m[i][current_row] / m[current_row][current_row];
        cout « "Eliminating element in row " « i + 1 « ", column " « current_row + 1 « ":" « endl;
        cout « "Multiplying row " « current_row + 1 « " by " « fixed « setprecision(4) « factor
             « " and subtracting from row " « i + 1 « "." « endl;
        m[i][current_row] = 0.0;
        for (int j = current_row + 1; j < total_cols; j++)
        {
            m[i][j] -= factor * m[current_row][j];
        }
        cout « endl;
    }
    return true;
}
```

#### 3.5.2.4 Exchange()

```
void Exchange (
              vector< vector< double > > & m,
              int row1,
              int row2)
```

```
{
    swap(m[row1], m[row2]);
    cout « "Swapping row " « row1 + 1 « " with row " « row2 + 1 « "." « endl;
}
```

#### 3.5.2.5 GaussianElimination()

```
int GaussianElimination (
              vector< vector< double > > & m,
              int rows,
              int cols)
```

```
{
    int exchange_count = 0;
    for (int i = 0; i < min(rows, cols - 1); i++)
    {
        cout « "Processing column " « i + 1 « "..." « endl;
        int imax = Pivoting(m, i, rows);
        if (imax != i)
        {
            Exchange(m, i, imax);
            exchange_count++;
        }
        else
        {
            cout « "No need to swap rows for column " « i + 1 « "." « endl;
        }

        // Check if pivot is zero
        if (fabs(m[i][i]) < 1e-12)
        {
            cout « "Warning: Pivot element in row " « i + 1 « " is close to zero. The matrix may be
      singular." « endl;
        }
        else
        {
            Eliminate(m, i, rows, cols);
        }

        // Display current matrix state with optimized formatting
        cout « "Current matrix state:" « endl;
        for (int r = 0; r < rows; r++)
        {
            for (int c = 0; c < cols; c++)
            {
                double coeff = round(m[r][c] * 1e12) / 1e12; // Handle floating-point precision
                if (fabs(coeff - round(coeff)) < 1e-12)
                {
                    cout « static_cast<long long>(round(coeff)) « "\t";
                }
                else
                {
                    cout « fixed « setprecision(2) « coeff « "\t";
                }
            }
            cout « endl;
        }
        cout « "--------------------------------------" « endl;
    }
    return exchange_count;
}
```

#### 3.5.2.6 IdentifyPivots()

```
vector< int > IdentifyPivots (
              const vector< vector< double > > & m,
            int rows,
            int cols)
{
    vector<int> pivots;
    for (int i = 0; i < min(rows, cols - 1); i++)
    {
        // Find the pivot in the current row
        int pivot_col = -1;
        for (int j = 0; j < cols - 1; j++)
        {
            if (fabs(m[i][j]) > 1e-12)
            {
                pivot_col = j;
                break;
            }
        }
        if (pivot_col != -1)
            pivots.push_back(pivot_col);
    }
    return pivots;
}
```

#### 3.5.2.7 Pivoting()

```
int Pivoting (
            const vector< vector< double > > & m,
            int current_row,
            int total_rows)
{
    int imax = current_row;
    double max_val = fabs(m[current_row][current_row]);
    for (int i = current_row + 1; i < total_rows; i++)
    {
        if (fabs(m[i][current_row]) > max_val)
        {
            imax = i;
            max_val = fabs(m[i][current_row]);
        }
    }
    return imax;
}
```

#### 3.5.2.8 ShowGeneralSolution()

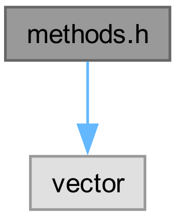
```
void ShowGeneralSolution (
            const vector< vector< double > > & m,
            int rows,
            int cols,
            int rank)
{
    cout « "The system has infinitely many solutions." « endl;
    cout « "Solution space dimension: " « (cols - 1 - rank) « endl;

    // Identify pivot columns
    vector<int> pivots = IdentifyPivots(m, rows, cols);

    // Identify free variables
    vector<int> free_vars;
    for (int j = 0; j < cols - 1; j++)
    {
        if (find(pivots.begin(), pivots.end(), j) == pivots.end())
        {
            free_vars.push_back(j);
        }
    }

    // Assign parameters to free variables
    int num_free = free_vars.size();
    vector<string> params;
    for (int i = 0; i < num_free; i++)
    {
        params.push_back("t" + to_string(i + 1));
    }

    // Initialize solution vector with parameters
    vector<double> particular_solution(cols - 1, 0.0);
    vector<vector<double» basis_vectors;

    // Find a particular solution by setting all free variables to 0
    for (int i = rows - 1; i >= 0; i--)
    {
        // Find the first non-zero coefficient in the row
        int pivot_col = -1;
        for (int j = 0; j < cols - 1; j++)
        {
            if (fabs(m[i][j]) > 1e-12)
            {
                pivot_col = j;
                break;
            }
        }

        if (pivot_col == -1)
        {
            continue; // 0 = 0, skip
        }

        double rhs = m[i][cols - 1];
        for (int j = pivot_col + 1; j < cols - 1; j++)
        {
            rhs -= m[i][j] * particular_solution[j];
        }
        particular_solution[pivot_col] = rhs / m[i][pivot_col];
    }

    // Now, find basis vectors by setting each free variable to 1 and others to 0
    for (int i = 0; i < num_free; i++)
    {
        vector<double> basis(cols - 1, 0.0);
        basis[free_vars[i]] = 1.0; // Set the free variable to 1

        // Perform back-substitution for pivot variables
        for (int r = rank - 1; r >= 0; r--)
        {
            int pivot_col = pivots[r];
            double rhs = 0.0;
            for (int j = pivot_col + 1; j < cols - 1; j++)
            {
                rhs -= m[r][j] * basis[j];
            }
            basis[pivot_col] = rhs / m[r][pivot_col];
        }

        basis_vectors.push_back(basis);
    }

    // Display the general solution
    cout « "General solution:" « endl;
    cout « "x = [";
    for (int j = 0; j < cols - 1; j++)
    {
        cout « fixed « setprecision(4) « particular_solution[j];
        if (j < cols - 2)
            cout « ", ";
    }
    cout « "]";

    for (int i = 0; i < num_free; i++)
    {
        cout « " + " « params[i] « " * [";
        for (int j = 0; j < cols - 1; j++)
        {
            cout « fixed « setprecision(4) « basis_vectors[i][j];
            if (j < cols - 2)
                cout « ", ";
        }
        cout « "]";
        if (i < num_free - 1)
            cout « " + ";
    }
    cout « endl
         « endl;
}
```

## 3.6 methods.h File Reference

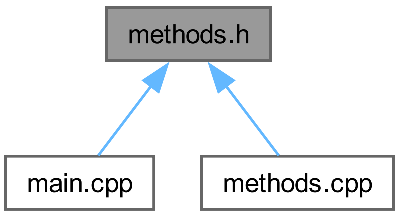
Core computational functions for solving linear systems.

```
#include <vector>
```

Include dependency graph for methods.h:



This graph shows which files directly or indirectly include this file:



**Functions**

- int GaussianElimination (std::vector< std::vector< double > > &m, int rows, int cols)

  *Performs Gaussian elimination on the matrix.*
- int DetermineRank (const std::vector< std::vector< double > > &m, int rows, int cols)

  *Determines the rank of the matrix.*
- bool BackSubstitution (const std::vector< std::vector< double > > &m, int rows, int cols, std::vector< double > &solution)

  *Performs back-substitution to find the unique solution.*
- void ShowGeneralSolution (const std::vector< std::vector< double > > &m, int rows, int cols, int rank)

  *Displays the general solution for systems with infinitely many solutions.*

### 3.6.1 Detailed Description

Core computational functions for solving linear systems.

**Author**

Gilbert Young

**Date**

2024/09/25

## 3.6.2 Function Documentation

### 3.6.2.1 BackSubstitution()

```
bool BackSubstitution (
            const std::vector< std::vector< double > > & m,
            int rows,
            int cols,
            std::vector< double > & solution)
```

Performs back-substitution to find the unique solution.

**Parameters**

| | |
|---|---|
| *m* | The upper triangular matrix after Gaussian elimination. |
| *rows* | Number of rows in the matrix. |
| *cols* | Number of columns in the matrix. |
| *solution* | Reference to store the solution vector. |

**Returns**

true If a unique solution exists.

false If the system is inconsistent.

### 3.6.2.2 DetermineRank()

```
int DetermineRank (
            const std::vector< std::vector< double > > & m,
            int rows,
            int cols)
```

Determines the rank of the matrix.

**Parameters**

| | |
|---|---|
| *m* | The matrix. |
| *rows* | Number of rows in the matrix. |
| *cols* | Number of columns in the matrix. |

**Returns**

int The rank of the matrix.

### 3.6.2.3 GaussianElimination()

```
int GaussianElimination (
            std::vector< std::vector< double > > & m,
            int rows,
            int cols)
```

Performs Gaussian elimination on the matrix.

**Parameters**

| | |
|---|---|
| *m* | Reference to the matrix to be modified. |
| *rows* | Number of rows in the matrix. |
| *cols* | Number of columns in the matrix. |

**Returns**

int Number of row exchanges performed.

### 3.6.2.4 ShowGeneralSolution()

```
void ShowGeneralSolution (
            const std::vector< std::vector< double > > & m,
            int rows,
            int cols,
            int rank)
```

Displays the general solution for systems with infinitely many solutions.

**Parameters**

| | |
|---|---|
| *m* | The matrix after Gaussian elimination. |
| *rows* | Number of rows in the matrix. |
| *cols* | Number of columns in the matrix. |
| *rank* | The rank of the matrix. |

## 3.7 methods.h

Go to the documentation of this file.

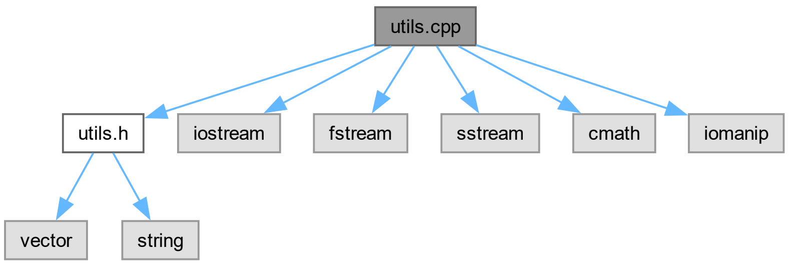
```
#ifndef METHODS_H
#define METHODS_H

#include <vector>

int GaussianElimination(std::vector<std::vector<double» &m, int rows, int cols);

int DetermineRank(const std::vector<std::vector<double» &m, int rows, int cols);

bool BackSubstitution(const std::vector<std::vector<double» &m, int rows, int cols,
     std::vector<double> &solution);

void ShowGeneralSolution(const std::vector<std::vector<double» &m, int rows, int cols, int rank);

#endif // METHODS_H
```

## 3.8 utils.cpp File Reference

Implementation of utility functions for matrix operations.

```
#include "utils.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
#include <iomanip>
```

Include dependency graph for utils.cpp:



**Functions**

- bool InitMatrix (vector< vector< double > > &m, const string &filename, int &rows, int &cols)
- void ShowEquations (const vector< vector< double > > &m, int rows, int cols)
- bool CheckConsistency (const vector< vector< double > > &m, int rows, int cols)
- void DisplaySolution (const vector< double > &solution)

### 3.8.1 Detailed Description

Implementation of utility functions for matrix operations.

**Author**

Gilbert Young

**Date**

2024/09/25

This file contains the implementations of functions that handle reading matrices from `.in` files and displaying the corresponding system of linear equations. These utility functions are essential for the initialization and output of matrix data used in solving linear systems.

## 3.8.2 Function Documentation

### 3.8.2.1 CheckConsistency()

```
bool CheckConsistency (
            const vector< vector< double > > & m,
            int rows,
            int cols)
```

```
{
    for (int i = 0; i < rows; i++)
    {
        bool all_zero = true;
        for (int j = 0; j < cols - 1; j++)
        {
            if (fabs(m[i][j]) > 1e-12)
            {
                all_zero = false;
                break;
            }
        }
        if (all_zero && fabs(m[i][cols - 1]) > 1e-12)
        {
            return false;
        }
    }
    return true;
}
```

### 3.8.2.2 DisplaySolution()

```
void DisplaySolution (
            const vector< double > & solution)
```

```
{
    cout « "The system has a unique solution:" « endl;
    for (size_t i = 0; i < solution.size(); i++)
    {
        cout « "x" « i + 1 « " = " « fixed « setprecision(4) « solution[i] « endl;
    }
}
```

### 3.8.2.3 InitMatrix()

```
bool InitMatrix (
            vector< vector< double > > & m,
            const string & filename,
            int & rows,
            int & cols)
```

```
{
    ifstream in(filename);
    if (!in.is_open())
    {
        cerr « "Error: Cannot open file " « filename « endl;
        return false;
    }

    // Read the matrix dimensions dynamically
    string line;
    rows = 0;
    cols = 0;
    vector<vector<double» temp_matrix;
    while (getline(in, line))
    {
        if (line.empty())
            continue; // Skip empty lines
        vector<double> row;
        double num;
        istringstream iss(line);
        while (iss » num)
        {
            row.push_back(num);
        }
        if (cols == 0)
        {
            cols = row.size();
        }
        else if ((int)row.size() != cols)
        {
            cerr « "Error: Inconsistent number of columns in the file." « endl;
            in.close();
            return false;
        }
        temp_matrix.push_back(row);
        rows++;
    }
    in.close();

    if (rows == 0 || cols < 2)
    {
        cerr « "Error: The matrix must have at least one equation and one variable." « endl;
        return false;
    }

    // Assign to m
    m = temp_matrix;
    return true;
}
```

#### 3.8.2.4 ShowEquations()

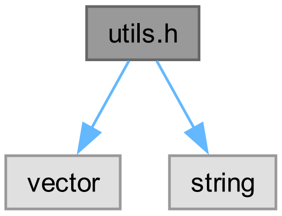
```
void ShowEquations (
               const vector< vector< double > > & m,
               int rows,
               int cols)
{
    cout « "The current system of linear equations is:" « endl;
    for (int i = 0; i < rows; i++)
    {
        string equation = "";
        for (int j = 0; j < cols - 1; j++)
        {
            // Check if the coefficient is an integer
            double coeff = round(m[i][j] * 1e12) / 1e12; // Handle floating-point precision
            if (fabs(coeff - round(coeff)) < 1e-12)
            {
                equation += to_string(static_cast<long long>(round(coeff))) + "x" + to_string(j + 1);
            }
            else
            {
                equation += to_string(round(m[i][j] * 10000) / 10000.0) + "x" + to_string(j + 1);
            }

            if (j < cols - 2)
                equation += " + ";
        }
        // Handle constant term
        double const_term = round(m[i][cols - 1] * 1e12) / 1e12;
        if (fabs(const_term - round(const_term)) < 1e-12)
        {
            equation += " = " + to_string(static_cast<long long>(round(const_term)));
        }
        else
        {
            equation += " = " + to_string(round(m[i][cols - 1] * 10000) / 10000.0);
        }

        cout « equation « endl;
    }
    cout « endl;
}
```

## 3.9 utils.h File Reference

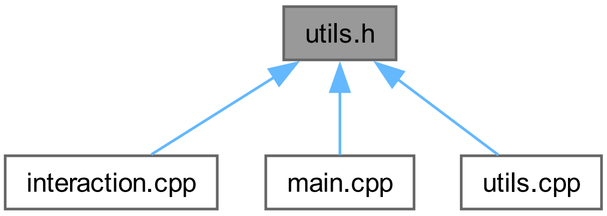
Utility functions for matrix initialization and display.

```
#include <vector>
#include <string>
```

Include dependency graph for utils.h:



This graph shows which files directly or indirectly include this file:



## Functions

- bool InitMatrix (std::vector< std::vector< double > > &m, const std::string &filename, int &rows, int &cols)

    *Initializes the matrix by reading from a .in file.*
- void ShowEquations (const std::vector< std::vector< double > > &m, int rows, int cols)

    *Displays the system of linear equations.*
- bool CheckConsistency (const std::vector< std::vector< double > > &m, int rows, int cols)

    *Checks the consistency of the system of equations.*
- void DisplaySolution (const std::vector< double > &solution)

    *Displays the unique solution.*

### 3.9.1 Detailed Description

Utility functions for matrix initialization and display.

**Author**

Gilbert Young

**Date**

2024/09/25

## 3.9.2 Function Documentation

### 3.9.2.1 CheckConsistency()

```
bool CheckConsistency (
             const std::vector< std::vector< double > > & m,
             int rows,
             int cols)
```

Checks the consistency of the system of equations.

**Parameters**

| | |
|---|---|
| *m* | The matrix representing the system. |
| *rows* | Number of rows in the matrix. |
| *cols* | Number of columns in the matrix. |

**Returns**

true If the system is consistent.

false If the system is inconsistent.

### 3.9.2.2 DisplaySolution()

```
void DisplaySolution (
             const std::vector< double > & solution)
```

Displays the unique solution.

**Parameters**

| | |
|---|---|
| *solution* | The solution vector. |

### 3.9.2.3 InitMatrix()

```
bool InitMatrix (
             std::vector< std::vector< double > > & m,
             const std::string & filename,
             int & rows,
             int & cols)
```

Initializes the matrix by reading from a .in file.

**Parameters**

| | |
|---|---|
| *m* | Reference to the matrix to be initialized. |
| *filename* | Name of the input file. |
| *rows* | Reference to store the number of rows. |
| *cols* | Reference to store the number of columns. |

**Returns**

true If the matrix was successfully initialized.

false If there was an error during initialization.

#### 3.9.2.4 ShowEquations()

```
void ShowEquations (
            const std::vector< std::vector< double > > & m,
            int rows,
            int cols)
```

Displays the system of linear equations.

**Parameters**

| | |
|---|---|
| *m* | The matrix representing the system. |
| *rows* | Number of equations. |
| *cols* | Number of variables plus one (for constants). |

## 3.10 utils.h

Go to the documentation of this file.

```
#ifndef UTILS_H
#define UTILS_H

#include <vector>
#include <string>

bool InitMatrix(std::vector<std::vector<double» &m, const std::string &filename, int &rows, int
     &cols);

void ShowEquations(const std::vector<std::vector<double» &m, int rows, int cols);

bool CheckConsistency(const std::vector<std::vector<double» &m, int rows, int cols);

void DisplaySolution(const std::vector<double> &solution);

#endif // UTILS_H
```

# Index