

# Gaussian Elimination Solver

1.0

Generated by Doxygen 1.12.0



<b>1 Gaussian Elimination Solver</b>	<b>1</b>
1.0.1 Features	1
1.0.2 Usage	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 File Documentation</b>	<b>5</b>
3.1 interaction.cpp File Reference	5
3.1.1 Detailed Description	6
3.1.2 Function Documentation	6
3.1.2.1 AskRunAgain()	6
3.1.2.2 SelectInputFile()	6
3.1.2.3 WaitForExit()	7
3.2 interaction.h File Reference	7
3.2.1 Detailed Description	8
3.2.2 Function Documentation	9
3.2.2.1 AskRunAgain()	9
3.2.2.2 SelectInputFile()	9
3.2.2.3 WaitForExit()	10
3.3 interaction.h	10
3.4 main.cpp File Reference	10
3.4.1 Detailed Description	11
3.4.2 Function Documentation	11
3.4.2.1 main()	11
3.5 methods.cpp File Reference	12
3.5.1 Detailed Description	13
3.5.2 Function Documentation	13
3.5.2.1 BackSubstitution()	13
3.5.2.2 DetermineRank()	14
3.5.2.3 Eliminate()	14
3.5.2.4 Exchange()	15
3.5.2.5 GaussianElimination()	15
3.5.2.6 IdentifyPivots()	15
3.5.2.7 Pivoting()	16
3.5.2.8 ShowGeneralSolution()	16
3.6 methods.h File Reference	17
3.6.1 Detailed Description	18
3.6.2 Function Documentation	19
3.6.2.1 BackSubstitution()	19
3.6.2.2 DetermineRank()	19
3.6.2.3 GaussianElimination()	19
3.6.2.4 ShowGeneralSolution()	20

---

3.7 methods.h . . . . .	20
3.8 utils.cpp File Reference . . . . .	21
3.8.1 Detailed Description . . . . .	21
3.8.2 Function Documentation . . . . .	22
3.8.2.1 CheckConsistency() . . . . .	22
3.8.2.2 DisplaySolution() . . . . .	22
3.8.2.3 InitMatrix() . . . . .	22
3.8.2.4 ShowEquations() . . . . .	23
3.9 utils.h File Reference . . . . .	23
3.9.1 Detailed Description . . . . .	24
3.9.2 Function Documentation . . . . .	25
3.9.2.1 CheckConsistency() . . . . .	25
3.9.2.2 DisplaySolution() . . . . .	25
3.9.2.3 InitMatrix() . . . . .	25
3.9.2.4 ShowEquations() . . . . .	26
3.10 utils.h . . . . .	26
<b>Index</b>	<b>27</b>

# 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:

<a href="#">interaction.cpp</a>	Implementation of user interaction functions . . . . .	5
<a href="#">interaction.h</a>	User interaction functions . . . . .	7
<a href="#">main.cpp</a>	Entry point for the Gaussian Elimination Solver project . . . . .	10
<a href="#">methods.cpp</a>	Implementation of computational functions for solving linear systems . . . . .	12
<a href="#">methods.h</a>	Core computational functions for solving linear systems . . . . .	17
<a href="#">utils.cpp</a>	Implementation of utility functions for matrix operations . . . . .	21
<a href="#">utils.h</a>	Utility functions for matrix initialization and display . . . . .	23





## Chapter 3

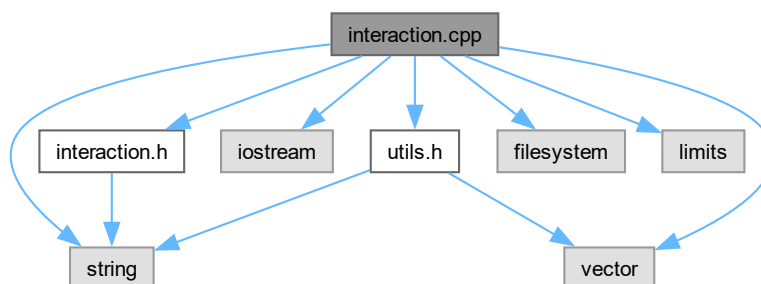
# File Documentation

### 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').

```
00091 {  
00092     char choice;  
00093     while (true)  
00094     {  
00095         cout << "\nDo you want to run the program again? (y/n): ";  
00096         cin >> choice;  
00097  
00098         if (choice == 'y' || choice == 'Y' || choice == 'n' || choice == 'N')  
00099         {  
00100             break;  
00101         }  
00102         else  
00103         {  
00104             cout << "Invalid input. Please enter 'y' or 'n'." << endl;  
00105         }  
00106     }  
00107     return choice;  
00108 }
```

#### 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.

```

00029 {
00030     vector<string> in_files;
00031     for (const auto &entry : filesystem::directory_iterator(filesystem::current_path()))
00032     {
00033         if (entry.is_regular_file())
00034         {
00035             string filename = entry.path().filename().string();
00036             if (filename.size() >= 3 && filename.substr(filename.size() - 3) == ".in")
00037             {
00038                 in_files.push_back(filename);
00039             }
00040         }
00041     }
00042
00043     string selected_file;
00044     if (in_files.empty())
00045     {
00046         cout << "No .in files found in the current directory." << endl;
00047         return "";
00048     }
00049     else if (in_files.size() == 1)
00050     {
00051         selected_file = in_files[0];
00052         cout << "Found one .in file: " << selected_file << " . Automatically selecting it." << endl;
00053     }
00054     else
00055     {
00056         cout << "Multiple .in files found. Please select one:" << endl;
00057         for (size_t i = 0; i < in_files.size(); i++)
00058         {
00059             cout << i + 1 << ". " << in_files[i] << endl;
00060         }
00061         int file_choice;
00062         // Improved input validation
00063         while (true)
00064         {
00065             cout << "Enter the number of the file you want to use (1-" << in_files.size() << "): ";
00066             cin >> file_choice;
00067
00068             if (cin.fail() || file_choice < 1 || file_choice > static_cast<int>(in_files.size()))
00069             {
00070                 cin.clear(); // Clear error flags
00071                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
00072                 cout << "Invalid input. Please enter a number between 1 and " << in_files.size() << ". " <<
endl;
00073             }
00074             else
00075             {
00076                 break;
00077             }
00078         }
00079         selected_file = in_files[file_choice - 1];
00080     }
00081     cout << endl;
00082     return selected_file;
00083 }

```

## 3.1.2.3 WaitForExit()

```
void WaitForExit ()
```

Waits for the user to press Enter before exiting.

```

00114 {
00115     cout << "\nPress Enter to exit...";
00116     cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
00117     cin.get(); // Wait for Enter key
00118 }

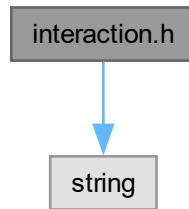
```

## 3.2 interaction.h File Reference

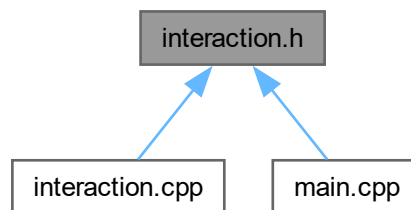
User interaction functions.

```
#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').

```
00091 {
00092     char choice;
00093     while (true)
00094     {
00095         cout << "\nDo you want to run the program again? (y/n): ";
00096         cin >> choice;
00097
00098         if (choice == 'y' || choice == 'Y' || choice == 'n' || choice == 'N')
00099         {
00100             break;
00101         }
00102         else
00103         {
00104             cout << "Invalid input. Please enter 'y' or 'n'." << endl;
00105         }
00106     }
00107     return choice;
00108 }
```

### 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.

```
00029 {
00030     vector<string> in_files;
00031     for (const auto &entry : filesystem::directory_iterator(filesystem::current_path()))
00032     {
00033         if (entry.is_regular_file())
00034         {
00035             string filename = entry.path().filename().string();
00036             if (filename.size() >= 3 && filename.substr(filename.size() - 3) == ".in")
00037             {
00038                 in_files.push_back(filename);
00039             }
00040         }
00041     }
00042
00043     string selected_file;
00044     if (in_files.empty())
00045     {
00046         cout << "No .in files found in the current directory." << endl;
00047         return "";
00048     }
00049     else if (in_files.size() == 1)
00050     {
00051         selected_file = in_files[0];
00052         cout << "Found one .in file: " << selected_file << " . Automatically selecting it." << endl;
00053     }
00054     else
00055     {
00056         cout << "Multiple .in files found. Please select one:" << endl;
00057         for (size_t i = 0; i < in_files.size(); i++)
00058         {
```

```

00059         cout << i + 1 << ". " << in_files[i] << endl;
00060     }
00061     int file_choice;
00062     // Improved input validation
00063     while (true)
00064     {
00065         cout << "Enter the number of the file you want to use (1-" << in_files.size() << "): ";
00066         cin >> file_choice;
00067
00068         if (cin.fail() || file_choice < 1 || file_choice > static_cast<int>(in_files.size()))
00069         {
00070             cin.clear(); // Clear error flags
00071             cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
00072             cout << "Invalid input. Please enter a number between 1 and " << in_files.size() << ". " <<
endl;
00073         }
00074         else
00075         {
00076             break;
00077         }
00078     }
00079     selected_file = in_files[file_choice - 1];
00080 }
00081 cout << endl;
00082 return selected_file;
00083 }

```

### 3.2.2.3 WaitForExit()

```
void WaitForExit ()
```

Waits for the user to press Enter before exiting.

```

00114 {
00115     cout << "\nPress Enter to exit...";
00116     cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
00117     cin.get(); // Wait for Enter key
00118 }

```

## 3.3 interaction.h

[Go to the documentation of this file.](#)

```

00001
00008 #ifndef INTERACTION_H
00009 #define INTERACTION_H
00010
00011 #include <string>
00012
00018 std::string SelectInputFile();
00019
00025 char AskRunAgain();
00026
00030 void WaitForExit();
00031
00032 #endif // INTERACTION_H

```

## 3.4 main.cpp File Reference

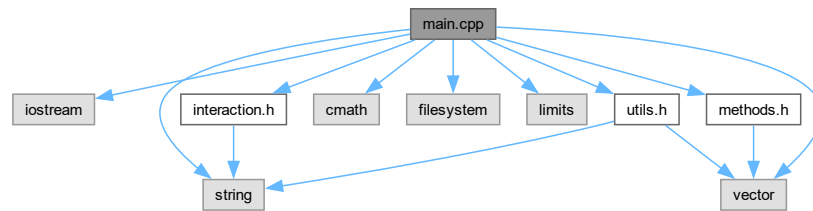
Entry point for the Gaussian Elimination Solver project.

```

#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 ()
00042 {
00043     char choice;
00044     do
00045     {
00046         string selected_file = SelectInputFile();
00047         if (selected_file.empty())
00048         {
00049             return 1; // File selection failed
00050         }
00051
00052         vector<vector<double>> matrix;
00053         int rows, cols;
00054         if (!InitMatrix(matrix, selected_file, rows, cols))
00055         {
00056             return 1; // Matrix initialization failed
00057         }
00058
00059         ShowEquations(matrix, rows, cols);
00060         cout << "Starting Gaussian elimination process..." << endl;
00061         int exchange_count = GaussianElimination(matrix, rows, cols);
00062         cout << "Gaussian elimination completed." << endl
00063              << endl;
```

```

00064
00065     int rank = DetermineRank(matrix, rows, cols);
00066     bool consistent = CheckConsistency(matrix, rows, cols);
00067
00068     if (!consistent)
00069     {
00070         cout << "The system of equations is inconsistent and has no solution." << endl;
00071     }
00072     else if (rank < (cols - 1))
00073     {
00074         ShowGeneralSolution(matrix, rows, cols, rank);
00075     }
00076     else
00077     {
00078         vector<double> solution;
00079         bool solvable = BackSubstitution(matrix, rows, cols, solution);
00080         if (solvable)
00081         {
00082             DisplaySolution(solution);
00083         }
00084         else
00085         {
00086             cout << "The system of equations is inconsistent and has no solution." << endl;
00087         }
00088     }
00089
00090     choice = AskRunAgain();
00091
00092     } while (choice == 'y' || choice == 'Y');
00093
00094     WaitForExit();
00095     return 0;
00096 }

```

### 3.5 methods.cpp File Reference

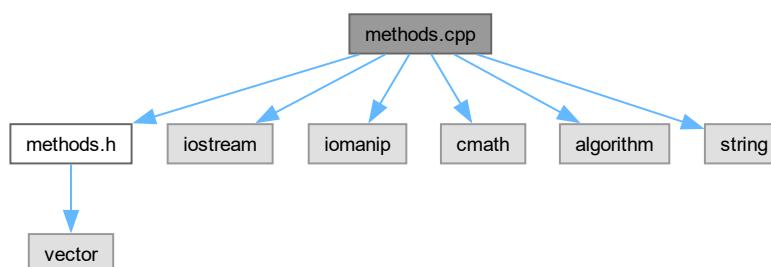
Implementation of computational functions for solving linear systems.

```

#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)
00120 {
00121     solution.assign(cols - 1, 0.0);
00122     cout << "Starting back-substitution process..." << endl;
00123     for (int i = rows - 1; i >= 0; i--)
00124     {
00125         // Find the first non-zero coefficient in the row
00126         int pivot_col = -1;
00127         for (int j = 0; j < cols - 1; j++)
00128         {
00129             if (fabs(m[i][j]) > 1e-12)
00130             {
00131                 pivot_col = j;
00132                 break;
00133             }
00134         }
00135         if (pivot_col == -1)
00136         {
00137             if (fabs(m[i][cols - 1]) > 1e-12)
00138             {
00139                 // Inconsistent equation
00140                 return false;
00141             }
00142             else
00143             {
00144                 // 0 = 0, skip
00145                 continue;
00146             }
00147         }
00148     }
00149 }
```

```

00150         double rhs = m[i][cols - 1];
00151         cout << "Calculating x" << pivot_col + 1 << ":" << endl;
00152         for (int j = pivot_col + 1; j < cols - 1; j++)
00153         {
00154             cout << "      " << fixed << setprecision(4) << m[i][j] << " * x" << j + 1
00155                  << " = " << m[i][j] * solution[j] << endl;
00156             rhs -= m[i][j] * solution[j];
00157         }
00158         cout << "      RHS after subtraction = " << rhs << endl;
00159         solution[pivot_col] = rhs / m[i][pivot_col];
00160         cout << "      x" << pivot_col + 1 << " = " << rhs << " / " << m[i][pivot_col]
00161              << " = " << fixed << setprecision(4) << solution[pivot_col] << endl
00162              << endl;
00163     }
00164     return true;
00165 }

```

### 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)
00040 {
00041     swap(m[row1], m[row2]);
00042     cout << "Swapping row " << row1 + 1 << " with row " << row2 + 1 << "." << endl;
00043 }
```

### 3.5.2.5 GaussianElimination()

```
int GaussianElimination (
    vector< vector< double > > & m,
    int rows,
    int cols)
00070 {
00071     int exchange_count = 0;
00072     for (int i = 0; i < min(rows, cols - 1); i++)
00073     {
00074         cout << "Processing column " << i + 1 << "..." << endl;
00075         int imax = Pivoting(m, i, rows);
00076         if (imax != i)
00077         {
00078             Exchange(m, i, imax);
00079             exchange_count++;
00080         }
00081         else
00082         {
00083             cout << "No need to swap rows for column " << i + 1 << "." << endl;
00084         }
00085         // Check if pivot is zero
00086         if (fabs(m[i][i]) < 1e-12)
00087         {
00088             cout << "Warning: Pivot element in row " << i + 1 << " is close to zero. The matrix may be
00089             singular." << endl;
00090         }
00091         else
00092         {
00093             Eliminate(m, i, rows, cols);
00094         }
00095         // Display current matrix state with optimized formatting
00096         cout << "Current matrix state:" << endl;
00097         for (int r = 0; r < rows; r++)
00098         {
00099             for (int c = 0; c < cols; c++)
00100             {
00101                 double coeff = round(m[r][c] * 1e12) / 1e12; // Handle floating-point precision
00102                 if (fabs(coeff - round(coeff)) < 1e-12)
00103                 {
00104                     cout << static_cast<long long>(round(coeff)) << "\t";
00105                 }
00106                 else
00107                 {
00108                     cout << fixed << setprecision(2) << coeff << "\t";
00109                 }
00110             }
00111             cout << endl;
00112         }
00113         cout << "-----" << endl;
00114     }
00115     return exchange_count;
00116 }
00117 }
```

### 3.5.2.6 IdentifyPivots()

```
vector< int > IdentifyPivots (
    const vector< vector< double > > & m,
```

```

        int rows,
        int cols)
00188 {
00189     vector<int> pivots;
00190     for (int i = 0; i < min(rows, cols - 1); i++)
00191     {
00192         // Find the pivot in the current row
00193         int pivot_col = -1;
00194         for (int j = 0; j < cols - 1; j++)
00195         {
00196             if (fabs(m[i][j]) > 1e-12)
00197             {
00198                 pivot_col = j;
00199                 break;
00200             }
00201         }
00202         if (pivot_col != -1)
00203             pivots.push_back(pivot_col);
00204     }
00205     return pivots;
00206 }

```

### 3.5.2.7 Pivoting()

```

int Pivoting (
    const vector< vector< double > > & m,
    int current_row,
    int total_rows)
00024 {
00025     int imax = current_row;
00026     double max_val = fabs(m[current_row][current_row]);
00027     for (int i = current_row + 1; i < total_rows; i++)
00028     {
00029         if (fabs(m[i][current_row]) > max_val)
00030         {
00031             imax = i;
00032             max_val = fabs(m[i][current_row]);
00033         }
00034     }
00035     return imax;
00036 }

```

### 3.5.2.8 ShowGeneralSolution()

```

void ShowGeneralSolution (
    const vector< vector< double > > & m,
    int rows,
    int cols,
    int rank)
00209 {
00210     cout << "The system has infinitely many solutions." << endl;
00211     cout << "Solution space dimension: " << (cols - 1 - rank) << endl;
00212
00213     // Identify pivot columns
00214     vector<int> pivots = IdentifyPivots(m, rows, cols);
00215
00216     // Identify free variables
00217     vector<int> free_vars;
00218     for (int j = 0; j < cols - 1; j++)
00219     {
00220         if (find(pivots.begin(), pivots.end(), j) == pivots.end())
00221         {
00222             free_vars.push_back(j);
00223         }
00224     }
00225
00226     // Assign parameters to free variables
00227     int num_free = free_vars.size();
00228     vector<string> params;
00229     for (int i = 0; i < num_free; i++)
00230     {
00231         params.push_back("t" + to_string(i + 1));
00232     }
00233 }

```

```

00234 // Initialize solution vector with parameters
00235 vector<double> particular_solution(cols - 1, 0.0);
00236 vector<vector<double>> basis_vectors;
00237
00238 // Find a particular solution by setting all free variables to 0
00239 for (int i = rows - 1; i >= 0; i--)
00240 {
00241     // Find the first non-zero coefficient in the row
00242     int pivot_col = -1;
00243     for (int j = 0; j < cols - 1; j++)
00244     {
00245         if (fabs(m[i][j]) > 1e-12)
00246         {
00247             pivot_col = j;
00248             break;
00249         }
00250     }
00251
00252     if (pivot_col == -1)
00253     {
00254         continue; // 0 = 0, skip
00255     }
00256
00257     double rhs = m[i][cols - 1];
00258     for (int j = pivot_col + 1; j < cols - 1; j++)
00259     {
00260         rhs -= m[i][j] * particular_solution[j];
00261     }
00262     particular_solution[pivot_col] = rhs / m[i][pivot_col];
00263 }
00264
00265 // Now, find basis vectors by setting each free variable to 1 and others to 0
00266 for (int i = 0; i < num_free; i++)
00267 {
00268     vector<double> basis(cols - 1, 0.0);
00269     basis[free_vars[i]] = 1.0; // Set the free variable to 1
00270
00271     // Perform back-substitution for pivot variables
00272     for (int r = rank - 1; r >= 0; r--)
00273     {
00274         int pivot_col = pivots[r];
00275         double rhs = 0.0;
00276         for (int j = pivot_col + 1; j < cols - 1; j++)
00277         {
00278             rhs -= m[r][j] * basis[j];
00279         }
00280         basis[pivot_col] = rhs / m[r][pivot_col];
00281     }
00282
00283     basis_vectors.push_back(basis);
00284 }
00285
00286 // Display the general solution
00287 cout << "General solution:" << endl;
00288 cout << "x = [";
00289 for (int j = 0; j < cols - 1; j++)
00290 {
00291     cout << fixed << setprecision(4) << particular_solution[j];
00292     if (j < cols - 2)
00293         cout << ", ";
00294 }
00295 cout << "];";
00296
00297 for (int i = 0; i < num_free; i++)
00298 {
00299     cout << " + " << params[i] << " * [";
00300     for (int j = 0; j < cols - 1; j++)
00301     {
00302         cout << fixed << setprecision(4) << basis_vectors[i][j];
00303         if (j < cols - 2)
00304             cout << ", ";
00305     }
00306     cout << "];";
00307     if (i < num_free - 1)
00308         cout << " + ";
00309 }
00310 cout << endl;
00311 << endl;
00312 }

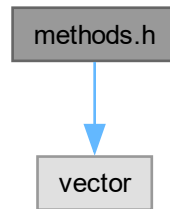
```

## 3.6 methods.h File Reference

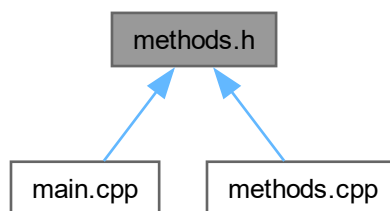
Core computational functions for solving linear systems.

```
#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**

<i>m</i>	The upper triangular matrix after Gaussian elimination.
<i>rows</i>	Number of rows in the matrix.
<i>cols</i>	Number of columns in the matrix.
<i>solution</i>	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**

<i>m</i>	The matrix.
<i>rows</i>	Number of rows in the matrix.
<i>cols</i>	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**

<i>m</i>	Reference to the matrix to be modified.
<i>rows</i>	Number of rows in the matrix.
<i>cols</i>	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**

<i>m</i>	The matrix after Gaussian elimination.
<i>rows</i>	Number of rows in the matrix.
<i>cols</i>	Number of columns in the matrix.
<i>rank</i>	The rank of the matrix.

**3.7 methods.h**

[Go to the documentation of this file.](#)

```
00001
00008 #ifndef METHODS_H
00009 #define METHODS_H
00010
00011 #include <vector>
00012
00021 int GaussianElimination(std::vector<std::vector<double>> &m, int rows, int cols);
00022
00031 int DetermineRank(const std::vector<std::vector<double>> &m, int rows, int cols);
00032
00043 bool BackSubstitution(const std::vector<std::vector<double>> &m, int rows, int cols,
00044                       std::vector<double> &solution);
00044
00053 void ShowGeneralSolution(const std::vector<std::vector<double>> &m, int rows, int cols, int rank);
00054
00055 #endif // METHODS_H
```

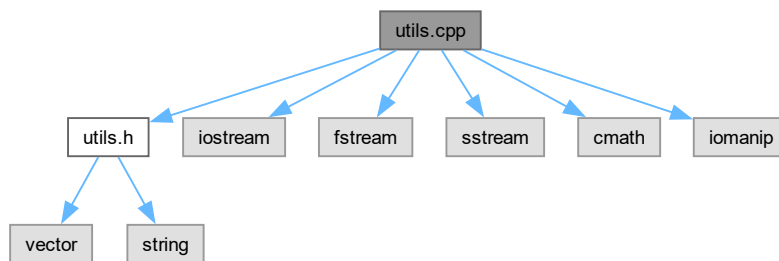


## 3.8 utils.cpp File Reference

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)
00113 {
00114     for (int i = 0; i < rows; i++)
00115     {
00116         bool all_zero = true;
00117         for (int j = 0; j < cols - 1; j++)
00118         {
00119             if (fabs(m[i][j]) > 1e-12)
00120             {
00121                 all_zero = false;
00122                 break;
00123             }
00124         }
00125         if (all_zero && fabs(m[i][cols - 1]) > 1e-12)
00126         {
00127             return false;
00128         }
00129     }
00130     return true;
00131 }

```

### 3.8.2.2 DisplaySolution()

```

void DisplaySolution (
    const vector< double > & solution)
00134 {
00135     cout << "The system has a unique solution:" << endl;
00136     for (size_t i = 0; i < solution.size(); i++)
00137     {
00138         cout << "x" << i + 1 << " = " << fixed << setprecision(4) << solution[i] << endl;
00139     }
00140 }

```

### 3.8.2.3 InitMatrix()

```

bool InitMatrix (
    vector< vector< double > > & m,
    const string & filename,
    int & rows,
    int & cols)
00024 {
00025     ifstream in(filename);
00026     if (!in.is_open())
00027     {
00028         cerr << "Error: Cannot open file " << filename << endl;
00029         return false;
00030     }
00031
00032     // Read the matrix dimensions dynamically
00033     string line;
00034     rows = 0;
00035     cols = 0;
00036     vector<vector<double>> temp_matrix;
00037     while (getline(in, line))
00038     {
00039         if (line.empty())
00040             continue; // Skip empty lines
00041         vector<double> row;
00042         double num;
00043         istringstream iss(line);
00044         while (iss >> num)
00045         {
00046             row.push_back(num);

```

```

00047     }
00048     if (cols == 0)
00049     {
00050         cols = row.size();
00051     }
00052     else if ((int)row.size() != cols)
00053     {
00054         cerr << "Error: Inconsistent number of columns in the file." << endl;
00055         in.close();
00056         return false;
00057     }
00058     temp_matrix.push_back(row);
00059     rows++;
00060 }
00061 in.close();
00062
00063 if (rows == 0 || cols < 2)
00064 {
00065     cerr << "Error: The matrix must have at least one equation and one variable." << endl;
00066     return false;
00067 }
00068
00069 // Assign to m
00070 m = temp_matrix;
00071 return true;
00072 }

```

### 3.8.2.4 ShowEquations()

```

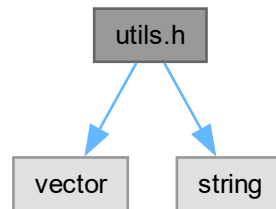
void ShowEquations (
    const vector< vector< double > > & m,
    int rows,
    int cols)
{
00075 {
00076     cout << "The current system of linear equations is:" << endl;
00077     for (int i = 0; i < rows; i++)
00078     {
00079         string equation = "";
00080         for (int j = 0; j < cols - 1; j++)
00081         {
00082             // Check if the coefficient is an integer
00083             double coeff = round(m[i][j] * 1e12) / 1e12; // Handle floating-point precision
00084             if (fabs(coeff - round(coeff)) < 1e-12)
00085             {
00086                 equation += to_string(static_cast<long long>(round(coeff))) + "x" + to_string(j + 1);
00087             }
00088             else
00089             {
00090                 equation += to_string(round(m[i][j] * 10000) / 10000.0) + "x" + to_string(j + 1);
00091             }
00092
00093             if (j < cols - 2)
00094                 equation += " + ";
00095         }
00096         // Handle constant term
00097         double const_term = round(m[i][cols - 1] * 1e12) / 1e12;
00098         if (fabs(const_term - round(const_term)) < 1e-12)
00099         {
00100             equation += " = " + to_string(static_cast<long long>(round(const_term)));
00101         }
00102         else
00103         {
00104             equation += " = " + to_string(round(m[i][cols - 1] * 10000) / 10000.0);
00105         }
00106         cout << equation << endl;
00107     }
00108     cout << endl;
00109 }
00110 }

```

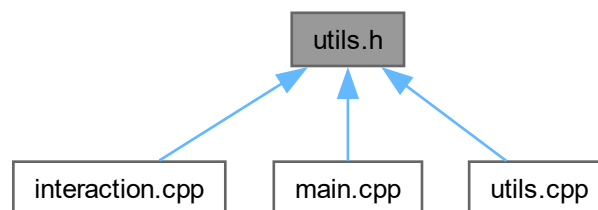
## 3.9 utils.h File Reference

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**

<i>m</i>	The matrix representing the system.
<i>rows</i>	Number of rows in the matrix.
<i>cols</i>	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**

<i>solution</i>	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**

<i>m</i>	Reference to the matrix to be initialized.
<i>filename</i>	Name of the input file.
<i>rows</i>	Reference to store the number of rows.
<i>cols</i>	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**

<i>m</i>	The matrix representing the system.
<i>rows</i>	Number of equations.
<i>cols</i>	Number of variables plus one (for constants).

**3.10 utils.h**

[Go to the documentation of this file.](#)

```
00001
00008 #ifndef UTILS_H
00009 #define UTILS_H
00010
00011 #include <vector>
00012 #include <string>
00013
00024 bool InitMatrix(std::vector<std::vector<double>> &m, const std::string &filename, int &rows, int
    &cols);
00025
00033 void ShowEquations(const std::vector<std::vector<double>> &m, int rows, int cols);
00034
00044 bool CheckConsistency(const std::vector<std::vector<double>> &m, int rows, int cols);
00045
00051 void DisplaySolution(const std::vector<double> &solution);
00052
00053 #endif // UTILS_H
```

# Index

- AskRunAgain
  - interaction.cpp, [6](#)
  - interaction.h, [9](#)
- BackSubstitution
  - methods.cpp, [13](#)
  - methods.h, [19](#)
- CheckConsistency
  - utils.cpp, [22](#)
  - utils.h, [25](#)
- DetermineRank
  - methods.cpp, [14](#)
  - methods.h, [19](#)
- DisplaySolution
  - utils.cpp, [22](#)
  - utils.h, [25](#)
- Eliminate
  - methods.cpp, [14](#)
- Exchange
  - methods.cpp, [14](#)
- Gaussian Elimination Solver, [1](#)
- GaussianElimination
  - methods.cpp, [15](#)
  - methods.h, [19](#)
- IdentifyPivots
  - methods.cpp, [15](#)
- InitMatrix
  - utils.cpp, [22](#)
  - utils.h, [25](#)
- interaction.cpp, [5](#)
  - AskRunAgain, [6](#)
  - SelectInputFile, [6](#)
  - WaitForExit, [7](#)
- interaction.h, [7](#)
  - AskRunAgain, [9](#)
  - SelectInputFile, [9](#)
  - WaitForExit, [10](#)
- main
  - main.cpp, [11](#)
- main.cpp, [10](#)
  - main, [11](#)
- methods.cpp, [12](#)
  - BackSubstitution, [13](#)
  - DetermineRank, [14](#)
  - Eliminate, [14](#)
  - Exchange, [14](#)
  - GaussianElimination, [15](#)
  - IdentifyPivots, [15](#)
  - Pivoting, [16](#)
  - ShowGeneralSolution, [16](#)
- methods.h, [17](#)
  - BackSubstitution, [19](#)
  - DetermineRank, [19](#)
  - GaussianElimination, [19](#)
  - ShowGeneralSolution, [20](#)
- Pivoting
  - methods.cpp, [16](#)
- SelectInputFile
  - interaction.cpp, [6](#)
  - interaction.h, [9](#)
- ShowEquations
  - utils.cpp, [23](#)
  - utils.h, [26](#)
- ShowGeneralSolution
  - methods.cpp, [16](#)
  - methods.h, [20](#)
- utils.cpp, [21](#)
  - CheckConsistency, [22](#)
  - DisplaySolution, [22](#)
  - InitMatrix, [22](#)
  - ShowEquations, [23](#)
- utils.h, [23](#)
  - CheckConsistency, [25](#)
  - DisplaySolution, [25](#)
  - InitMatrix, [25](#)
  - ShowEquations, [26](#)
- WaitForExit
  - interaction.cpp, [7](#)
  - interaction.h, [10](#)