Linear Equations Solver

1.0

Generated by Doxygen 1.12.0

Chapter 1

Linear Equations Solver

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

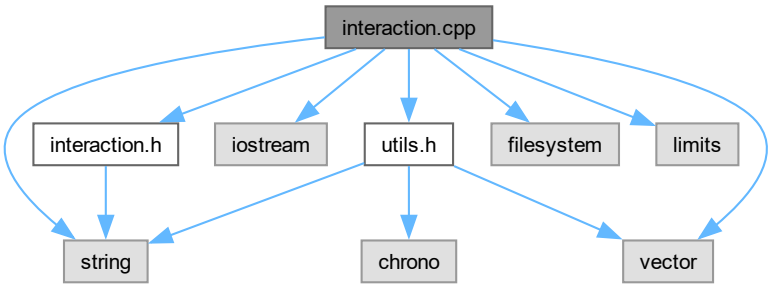
File Documentation

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.Returns an empty string if no file is selected.
- char AskRunAgain ()

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

Definition in file interaction.cpp.

3.1.2 Function Documentation

3.1.2.1 AskRunAgain()

```
char AskRunAgain ()
```

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

Definition at line 82 of file interaction.cpp.

```
00083 {
00084     char choice;
00085     while (true)
00086     {
00087         cout « "\nDo you want to run the program again? (y/n): ";
00088         cin » choice;
00089 
00090         if (choice == 'y' || choice == 'Y' || choice == 'n' || choice == 'N')
00091         {
00092             break;
00093         }
00094         else
00095         {
00096             cout « "Invalid input. Please enter 'y' or 'n'." « endl;
00097         }
00098     }
00099     return choice;
00100 }
```

3.1.2.2 SelectInputFile()

```
string SelectInputFile ()
```

Allows the user to select an input `.in` file from the current directory.Returns an empty string if no file is selected.

Definition at line 24 of file interaction.cpp.

```
00025 {
00026     vector<string> in_files;
00027     for (const auto &entry : filesystem::directory_iterator(filesystem::current_path()))
00028     {
00029         if (entry.is_regular_file())
00030         {
00031             string filename = entry.path().filename().string();
00032             if (filename.size() >= 3 && filename.substr(filename.size() - 3) == ".in")
```

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

3.1.2.3 WaitForExit()

```
void WaitForExit ()
```

Waits for the user to press Enter before exiting.

Definition at line 103 of file interaction.cpp.

```
00104 {
00105     cout « "\nPress Enter to exit...";
00106     cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
00107     cin.get();                                           // Wait for Enter key
00108 }
```

3.2 interaction.cpp

Go to the documentation of this file.

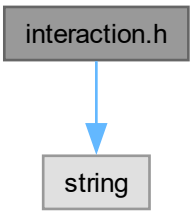
```
00001
00013 #include "interaction.h"
00014 #include <iostream>
00015 #include <vector>
00016 #include <string>
00017 #include <filesystem>
00018 #include <limits>
00019 #include "utils.h"
00020
00021 using namespace std;
```

```
00022
00024 string SelectInputFile()
00025 {
00026     vector<string> in_files;
00027     for (const auto &entry : filesystem::directory_iterator(filesystem::current_path()))
00028     {
00029         if (entry.is_regular_file())
00030         {
00031             string filename = entry.path().filename().string();
00032             if (filename.size() >= 3 && filename.substr(filename.size() - 3) == ".in")
00033             {
00034                 in_files.push_back(filename);
00035             }
00036         }
00037     }
00038
00039     string selected_file;
00040     if (in_files.empty())
00041     {
00042         cout « "No .in files found in the current directory." « endl;
00043         return "";
00044     }
00045     else if (in_files.size() == 1)
00046     {
00047         selected_file = in_files[0];
00048         cout « "Found one .in file: " « selected_file « ". Automatically selecting it." « endl;
00049     }
00050     else
00051     {
00052         cout « "Multiple .in files found. Please select one:" « endl;
00053         for (size_t i = 0; i < in_files.size(); i++)
00054         {
00055             cout « i + 1 « ". " « in_files[i] « endl;
00056         }
00057         int file_choice;
00058         // Improved input validation
00059         while (true)
00060         {
00061             cout « "Enter the number of the file you want to use (1-" « in_files.size() « "): ";
00062             cin » file_choice;
00063
00064             if (cin.fail() || file_choice < 1 || file_choice > static_cast<int>(in_files.size()))
00065             {
00066                 cin.clear();                                          // Clear error flags
00067                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
00068                 cout « "Invalid input. Please enter a number between 1 and " « in_files.size() « "." «
      endl;
00069             }
00070             else
00071             {
00072                 break;
00073             }
00074         }
00075         selected_file = in_files[file_choice - 1];
00076     }
00077     cout « endl;
00078     return selected_file;
00079 }
00080
00082 char AskRunAgain()
00083 {
00084     char choice;
00085     while (true)
00086     {
00087         cout « "\nDo you want to run the program again? (y/n): ";
00088         cin » choice;
00089
00090         if (choice == 'y' || choice == 'Y' || choice == 'n' || choice == 'N')
00091         {
00092             break;
00093         }
00094         else
00095         {
00096             cout « "Invalid input. Please enter 'y' or 'n'." « endl;
00097         }
00098     }
00099     return choice;
00100 }
00101
00103 void WaitForExit()
00104 {
00105     cout « "\nPress Enter to exit...";
00106     cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
00107     cin.get();                                           // Wait for Enter key
00108 }
```

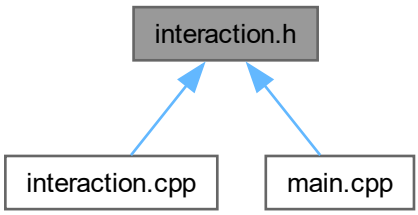
3.3 interaction.h File Reference

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.Returns an empty string if no file is selected.
- char AskRunAgain ()

  Return char The user's choice ('y', 'Y', 'n', 'N').
- void WaitForExit ()

  Waits for the user to press Enter before exiting.

3.3.1 Detailed Description

User interaction functions.

Author

Gilbert Young

Date

2024/09/25

Definition in file interaction.h.

3.3.2 Function Documentation

3.3.2.1 AskRunAgain()

```
char AskRunAgain ()
```

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

Definition at line 82 of file interaction.cpp.

```
00083 {
00084     char choice;
00085     while (true)
00086     {
00087         cout « "\nDo you want to run the program again? (y/n): ";
00088         cin » choice;
00089 
00090         if (choice == 'y' || choice == 'Y' || choice == 'n' || choice == 'N')
00091         {
00092             break;
00093         }
00094         else
00095         {
00096             cout « "Invalid input. Please enter 'y' or 'n'." « endl;
00097         }
00098     }
00099     return choice;
00100 }
```

3.3.2.2 SelectInputFile()

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

Allows the user to select an input `.in` file from the current directory.Returns an empty string if no file is selected.

Definition at line 24 of file interaction.cpp.

```
00025 {
00026     vector<string> in_files;
00027     for (const auto &entry : filesystem::directory_iterator(filesystem::current_path()))
00028     {
00029         if (entry.is_regular_file())
00030         {
00031             string filename = entry.path().filename().string();
00032             if (filename.size() >= 3 && filename.substr(filename.size() - 3) == ".in")
00033             {
00034                 in_files.push_back(filename);
00035             }
00036         }
00037     }
00038 
00039     string selected_file;
00040     if (in_files.empty())
00041     {
00042         cout « "No .in files found in the current directory." « endl;
00043         return "";
00044     }
00045     else if (in_files.size() == 1)
00046     {
```

```
00047         selected_file = in_files[0];
00048         cout « "Found one .in file: " « selected_file « ". Automatically selecting it." « endl;
00049     }
00050     else
00051     {
00052         cout « "Multiple .in files found. Please select one:" « endl;
00053         for (size_t i = 0; i < in_files.size(); i++)
00054         {
00055             cout « i + 1 « ". " « in_files[i] « endl;
00056         }
00057         int file_choice;
00058         // Improved input validation
00059         while (true)
00060         {
00061             cout « "Enter the number of the file you want to use (1-" « in_files.size() « "): ";
00062             cin » file_choice;
00063 
00064             if (cin.fail() || file_choice < 1 || file_choice > static_cast<int>(in_files.size()))
00065             {
00066                 cin.clear();                                          // Clear error flags
00067                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
00068                 cout « "Invalid input. Please enter a number between 1 and " « in_files.size() « "." «
      endl;
00069             }
00070             else
00071             {
00072                 break;
00073             }
00074         }
00075         selected_file = in_files[file_choice - 1];
00076     }
00077     cout « endl;
00078     return selected_file;
00079 }
```

3.3.2.3 WaitForExit()

```
void WaitForExit ()
```

Waits for the user to press Enter before exiting.

Definition at line 103 of file interaction.cpp.

```
00104 {
00105     cout « "\nPress Enter to exit...";
00106     cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
00107     cin.get();                                           // Wait for Enter key
00108 }
```

3.4 interaction.h

Go to the documentation of this file.

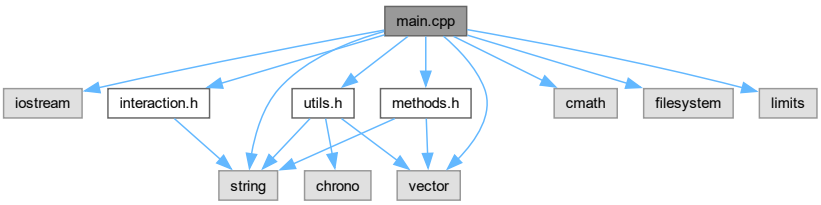
```
00001 
00008 #ifndef INTERACTION_H
00009 #define INTERACTION_H
00010 
00011 #include <string>
00012 
00013 std::string SelectInputFile();
00014 
00015 char AskRunAgain();
00016 
00017 void WaitForExit();
00018 
00019 #endif // INTERACTION_H
```

3.5 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 ()

3.5.1 Detailed Description

Entry point for the Gaussian Elimination Solver project.

Author

Gilbert Young

Date

2024/09/25

Definition in file main.cpp.

3.5.2 Function Documentation

3.5.2.1 main()

```
int main ()
```

Definition at line 41 of file main.cpp.

```
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         // Start timer after selecting the file
00053         auto start_time = StartTimer();
00054
00055         vector<vector<double» matrix;
00056         int rows, cols;
00057         if (!InitMatrix(matrix, selected_file, rows, cols))
00058         {
00059             return 1; // Matrix initialization failed
00060         }
00061
00062         ShowEquations(matrix, rows, cols);
00063         cout « "Starting Gaussian elimination process..." « endl;
00064         int exchange_count = GaussianElimination(matrix, rows, cols);
00065         cout « "Gaussian elimination completed." « endl
00066              « endl;
00067
00068         int rank = DetermineRank(matrix, rows, cols);
00069         bool consistent = CheckConsistency(matrix, rows, cols);
00070
00071         if (!consistent)
00072         {
00073             cout « "The system of equations is inconsistent and has no solution." « endl;
00074         }
00075         else if (rank < (cols - 1))
00076         {
00077             ShowGeneralSolution(matrix, rows, cols, rank);
00078         }
00079         else
00080         {
00081             vector<double> solution;
00082             bool solvable = BackSubstitution(matrix, rows, cols, solution);
00083             if (solvable)
00084             {
00085                 DisplaySolution(solution);
00086             }
00087             else
00088             {
00089                 cout « "The system of equations is inconsistent and has no solution." « endl;
00090             }
00091         }
00092
00093         // Stop timer after the solution is displayed
00094         StopTimer(start_time);
00095         choice = AskRunAgain();
00096
00097     } while (choice == 'y' || choice == 'Y');
00098
00099     WaitForExit();
00100     return 0;
00101 }
```

3.6 main.cpp

Go to the documentation of this file.

```
00001
00029 #include <iostream>
00030 #include <string>
00031 #include <vector>
00032 #include <cmath> // Included for fabs function
00033 #include <filesystem>
```

```
00034 #include <limits> // For std::numeric_limits
00035 #include "utils.h"
00036 #include "methods.h"
00037 #include "interaction.h"
00038
00039 using namespace std;
00040
00041 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         // Start timer after selecting the file
00053         auto start_time = StartTimer();
00054
00055         vector<vector<double» matrix;
00056         int rows, cols;
00057         if (!InitMatrix(matrix, selected_file, rows, cols))
00058         {
00059             return 1; // Matrix initialization failed
00060         }
00061
00062         ShowEquations(matrix, rows, cols);
00063         cout « "Starting Gaussian elimination process..." « endl;
00064         int exchange_count = GaussianElimination(matrix, rows, cols);
00065         cout « "Gaussian elimination completed." « endl
00066              « endl;
00067
00068         int rank = DetermineRank(matrix, rows, cols);
00069         bool consistent = CheckConsistency(matrix, rows, cols);
00070
00071         if (!consistent)
00072         {
00073             cout « "The system of equations is inconsistent and has no solution." « endl;
00074         }
00075         else if (rank < (cols - 1))
00076         {
00077             ShowGeneralSolution(matrix, rows, cols, rank);
00078         }
00079         else
00080         {
00081             vector<double> solution;
00082             bool solvable = BackSubstitution(matrix, rows, cols, solution);
00083             if (solvable)
00084             {
00085                 DisplaySolution(solution);
00086             }
00087             else
00088             {
00089                 cout « "The system of equations is inconsistent and has no solution." « endl;
00090             }
00091         }
00092
00093         // Stop timer after the solution is displayed
00094         StopTimer(start_time);
00095         choice = AskRunAgain();
00096
00097     } while (choice == 'y' || choice == 'Y');
00098
00099     WaitForExit();
00100     return 0;
00101 }
```

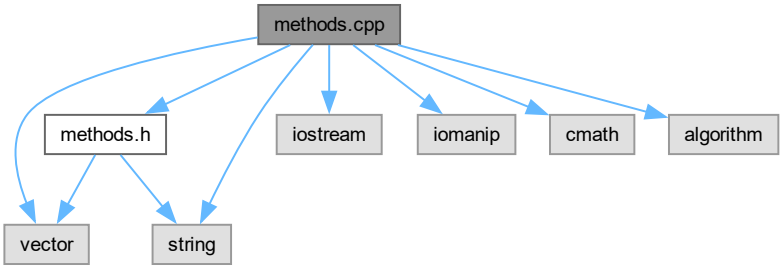
3.7 methods.cpp File Reference

Implementation of computational functions for solving linear systems.

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

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

Include dependency graph for methods.cpp:



Functions

- int Pivoting (const vector< vector< double > > &m, int current_row, int total_rows)

  Performs partial pivoting and returns the row index with the maximum pivot.
- void Exchange (vector< vector< double > > &m, int row1, int row2)

  Swaps two rows in the matrix and outputs the action.
- bool Eliminate (vector< vector< double > > &m, int current_row, int total_rows, int total_cols)

  Performs elimination on the matrix to form an upper triangular matrix.
- int GaussianElimination (vector< vector< double > > &m, int rows, int cols)

  Performs Gaussian elimination on the augmented matrix with partial pivoting.
- bool BackSubstitution (const vector< vector< double > > &m, int rows, int cols, vector< double > &solution)

  Performs back-substitution to find the solution vector.
- int DetermineRank (const vector< vector< double > > &m, int rows, int cols)

  Determines the rank of the coefficient matrix A (excluding augmented column).
- void ShowGeneralSolution (const vector< vector< double > > &m, int rows, int cols, int rank)

  Displays the general solution for systems with infinitely many solutions.
- vector< int > IdentifyPivots (const vector< vector< double > > &m, int rows, int cols)

  Identifies the pivot columns in the matrix.

3.7.1 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.

Definition in file methods.cpp.

3.7.2 Function Documentation

3.7.2.1 BackSubstitution()

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

Performs back-substitution to find the solution vector.

Parameters

m	The upper triangular matrix after Gaussian elimination.
rows	Number of rows in the matrix.
cols	Number of columns in the matrix (including augmented column).
solution	Reference to store the solution vector.

Returns

true If a unique solution exists.

false If the system is inconsistent.

Definition at line 137 of file methods.cpp.

```
00138 {
00139     solution.assign(cols - 1, 0.0);
00140     cout « "Starting back-substitution process..." « endl;
00141     for (int i = rows - 1; i >= 0; i--)
00142     {
00143         // Find the first non-zero coefficient in the row
00144         int pivot_col = -1;
00145         for (int j = 0; j < cols - 1; j++)
00146         {
00147             if (fabs(m[i][j]) > 1e-12)
00148             {
00149                 pivot_col = j;
00150                 break;
00151             }
00152         }
00153
00154         if (pivot_col == -1)
00155         {
00156             if (fabs(m[i][cols - 1]) > 1e-12)
00157             {
00158                 // Inconsistent equation
00159                 return false;
00160             }
00161             else
00162             {
00163                 // 0 = 0, skip
00164                 continue;
00165             }
00166         }
00167
00168         double rhs = m[i][cols - 1];
00169         cout « "Calculating x" « pivot_col + 1 « ":" « endl;
00170         for (int j = pivot_col + 1; j < cols - 1; j++)
00171         {
00172             cout « "    " « fixed « setprecision(4) « m[i][j] « " * x" « j + 1
00173                  « " = " « m[i][j] * solution[j] « endl;
00174             rhs -= m[i][j] * solution[j];
00175         }
00176         cout « "    RHS after subtraction = " « rhs « endl;
00177         solution[pivot_col] = rhs / m[i][pivot_col];
00178         cout « "    x" « pivot_col + 1 « " = " « rhs « " / " « m[i][pivot_col]
00179              « " = " « fixed « setprecision(4) « solution[pivot_col] « endl
00180              « endl;
00181     }
00182     return true;
00183 }
```

3.7.2.2 DetermineRank()

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

Determines the rank of the coefficient matrix A (excluding augmented column).

Parameters

m	The augmented matrix [A|b].
rows	Number of rows in the matrix.
cols	Number of columns in the matrix (including augmented column).

Returns

int The rank of the matrix A.

Definition at line 188 of file methods.cpp.

```
00189 {
00190     int rank = 0;
00191     for (int i = 0; i < rows; i++)
00192     {
00193         bool non_zero = false;
00194         for (int j = 0; j < cols - 1; j++)
00195         {
00196             if (fabs(m[i][j]) > 1e-12)
00197             {
00198                 non_zero = true;
00199                 break;
00200             }
00201         }
00202         if (non_zero)
00203             rank++;
00204     }
00205     return rank;
00206 }
```

3.7.2.3 Eliminate()

```
bool Eliminate (
            vector< vector< double > > & m,
            int current_row,
            int total_rows,
            int total_cols)
```

Performs elimination on the matrix to form an upper triangular matrix.

Definition at line 47 of file methods.cpp.

```
00048 {
00049     double pivot = m[current_row][current_row];
00050     if (fabs(pivot) < 1e-12)
00051     {
00052         // Pivot is too small, cannot eliminate
00053         return false;
00054     }
00055
00056     for (int i = current_row + 1; i < total_rows; i++)
00057     {
00058         double factor = m[i][current_row] / pivot;
00059         cout << "Eliminating element in row " << i + 1 << ", column " << current_row + 1 << ":" << endl;
00060         cout << "Multiplying row " << current_row + 1 << " by " << fixed << setprecision(4) << factor
00061              << " and subtracting from row " << i + 1 << "." << endl;
00062         m[i][current_row] = 0.0;
00063         for (int j = current_row + 1; j < total_cols; j++)
00064         {
00065             m[i][j] -= factor * m[current_row][j];
00066         }
00067         cout << endl;
00068     }
00069     return true;
00070 }
```

3.7.2.4 Exchange()

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

Swaps two rows in the matrix and outputs the action.

Definition at line 40 of file methods.cpp.

```
00041 {
00042     swap(m[row1], m[row2]);
00043     cout « "Swapping row " « row1 + 1 « " with row " « row2 + 1 « "." « endl;
00044 }
```

3.7.2.5 GaussianElimination()

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

Performs Gaussian elimination on the augmented matrix with partial pivoting.

Parameters

m	Reference to the augmented matrix [A|b] to be modified.
rows	Number of rows in the matrix.
cols	Number of columns in the matrix (including augmented column).

Returns

int Number of row exchanges performed during elimination.

Definition at line 75 of file methods.cpp.

```
00076 {
00077     int exchange_count = 0;
00078     int n = min(rows, cols - 1); // Number of variables
00079
00080     for (int k = 0; k < n; k++)
00081     {
00082         cout « "Processing column " « k + 1 « "..." « endl;
00083
00084         // Find the row with the maximum pivot element
00085         int imax = Pivoting(m, k, rows);
00086
00087         // Swap the current row with the pivot row if necessary
00088         if (imax != k)
00089         {
00090             Exchange(m, k, imax);
00091             exchange_count++;
00092         }
00093         else
00094         {
00095             cout « "No need to swap rows for column " « k + 1 « "." « endl;
00096         }
00097
00098         // Check if pivot element is near zero (singular matrix)
00099         if (fabs(m[k][k]) < 1e-12)
00100         {
00101             cout « "Warning: Pivot element in row " « k + 1 « " is close to zero. The matrix may be
    singular." « endl;
00102             continue; // Skip elimination for this pivot
00103         }
00104
```

```
00105         // Eliminate entries below the pivot
00106         if (!Eliminate(m, k, rows, cols))
00107         {
00108             cout « "Elimination failed for column " « k + 1 « "." « endl;
00109         }
00110 
00111         // Display current matrix state
00112         cout « "Current matrix state:" « endl;
00113         for (int r = 0; r < rows; r++)
00114         {
00115             for (int c = 0; c < cols; c++)
00116             {
00117                 double coeff = round(m[r][c] * 1e12) / 1e12; // Handle floating-point precision
00118                 if (fabs(coeff - round(coeff)) < 1e-12)
00119                 {
00120                     cout « static_cast<long long>(round(coeff)) « "\t";
00121                 }
00122                 else
00123                 {
00124                     cout « fixed « setprecision(2) « coeff « "\t";
00125                 }
00126             }
00127             cout « endl;
00128         }
00129         cout « "-------------------------------------" « endl;
00130     }
00131     return exchange_count;
00132 }
```

3.7.2.6 IdentifyPivots()

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

Identifies the pivot columns in the matrix.

Parameters

m	The matrix after Gaussian elimination.
rows	Number of rows in the matrix.
cols	Number of columns in the matrix (including augmented column).

Returns

std::vector<int> A vector containing the indices of the pivot columns.

Definition at line 320 of file methods.cpp.

```
00321 {
00322     vector<int> pivots;
00323     int n = min(rows, cols - 1);
00324     for (int i = 0; i < n; i++)
00325     {
00326         // Find the pivot column in the current row
00327         int pivot_col = -1;
00328         for (int j = 0; j < cols - 1; j++)
00329         {
00330             if (fabs(m[i][j]) > 1e-12)
00331             {
00332                 pivot_col = j;
00333                 break;
00334             }
00335         }
00336         if (pivot_col != -1)
00337             pivots.push_back(pivot_col);
00338     }
00339     return pivots;
00340 }
```

3.7.2.7 Pivoting()

```
int Pivoting (
            const vector< vector< double > > & m,
            int current_row,
            int total_rows)
```

Performs partial pivoting and returns the row index with the maximum pivot.

Definition at line 23 of file methods.cpp.

```
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         double val = fabs(m[i][current_row]);
00030         if (val > max_val)
00031         {
00032             imax = i;
00033             max_val = val;
00034         }
00035     }
00036     return imax;
00037 }
```

3.7.2.8 ShowGeneralSolution()

```
void ShowGeneralSolution (
            const vector< 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 (including augmented column).
rank	The rank of the coefficient matrix A.

Definition at line 211 of file methods.cpp.

```
00212 {
00213     cout « "The system has infinitely many solutions." « endl;
00214     cout « "Solution space dimension: " « (cols - 1 - rank) « endl;
00215
00216     // Identify pivot columns
00217     vector<int> pivots = IdentifyPivots(m, rows, cols);
00218
00219     // Identify free variables
00220     vector<int> free_vars;
00221     for (int j = 0; j < cols - 1; j++)
00222     {
00223         if (find(pivots.begin(), pivots.end(), j) == pivots.end())
00224         {
00225             free_vars.push_back(j);
00226         }
00227     }
00228
00229     // Assign parameters to free variables
00230     int num_free = free_vars.size();
00231     vector<string> params;
00232     for (int i = 0; i < num_free; i++)
00233     {
00234         params.push_back("t" + to_string(i + 1));
00235     }
```

```
00236
00237     // Initialize solution vector with parameters
00238     vector<double> particular_solution(cols - 1, 0.0);
00239     vector<vector<double» basis_vectors;
00240
00241     // Find a particular solution by setting all free variables to 0
00242     for (int i = rows - 1; i >= 0; i--)
00243     {
00244         // Find the first non-zero coefficient in the row
00245         int pivot_col = -1;
00246         for (int j = 0; j < cols - 1; j++)
00247         {
00248             if (fabs(m[i][j]) > 1e-12)
00249             {
00250                 pivot_col = j;
00251                 break;
00252             }
00253         }
00254
00255         if (pivot_col == -1)
00256         {
00257             continue; // 0 = 0, skip
00258         }
00259
00260         double rhs = m[i][cols - 1];
00261         for (int j = pivot_col + 1; j < cols - 1; j++)
00262         {
00263             rhs -= m[i][j] * particular_solution[j];
00264         }
00265         particular_solution[pivot_col] = rhs / m[i][pivot_col];
00266     }
00267
00268     // Now, find basis vectors by setting each free variable to 1 and others to 0
00269     for (int i = 0; i < num_free; i++)
00270     {
00271         vector<double> basis(cols - 1, 0.0);
00272         basis[free_vars[i]] = 1.0; // Set the free variable to 1
00273
00274         // Perform back-substitution for pivot variables
00275         for (int r = rank - 1; r >= 0; r--)
00276         {
00277             int pivot_col = pivots[r];
00278             double rhs = 0.0;
00279             for (int j = pivot_col + 1; j < cols - 1; j++)
00280             {
00281                 rhs -= m[r][j] * basis[j];
00282             }
00283             basis[pivot_col] = rhs / m[r][pivot_col];
00284         }
00285
00286         basis_vectors.push_back(basis);
00287     }
00288
00289     // Display the general solution
00290     cout « "General solution:" « endl;
00291     cout « "x = [";
00292     for (int j = 0; j < cols - 1; j++)
00293     {
00294         cout « fixed « setprecision(4) « particular_solution[j];
00295         if (j < cols - 2)
00296             cout « ", ";
00297     }
00298     cout « "]";
00299
00300     for (int i = 0; i < num_free; i++)
00301     {
00302         cout « " + " « params[i] « " * [";
00303         for (int j = 0; j < cols - 1; j++)
00304         {
00305             cout « fixed « setprecision(4) « basis_vectors[i][j];
00306             if (j < cols - 2)
00307                 cout « ", ";
00308         }
00309         cout « "]";
00310         if (i < num_free - 1)
00311             cout « " + ";
00312     }
00313     cout « endl
00314          « endl;
00315 }
```

3.8 methods.cpp

Go to the documentation of this file.

```
00001
00012 #include "methods.h"
00013 #include <iostream>
00014 #include <iomanip>
00015 #include <cmath>
00016 #include <algorithm>
00017 #include <string>
00018 #include <vector>
00019
00020 using namespace std;
00021
00023 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         double val = fabs(m[i][current_row]);
00030         if (val > max_val)
00031         {
00032             imax = i;
00033             max_val = val;
00034         }
00035     }
00036     return imax;
00037 }
00038
00040 void Exchange(vector<vector<double» &m, int row1, int row2)
00041 {
00042     swap(m[row1], m[row2]);
00043     cout « "Swapping row " « row1 + 1 « " with row " « row2 + 1 « "." « endl;
00044 }
00045
00047 bool Eliminate(vector<vector<double» &m, int current_row, int total_rows, int total_cols)
00048 {
00049     double pivot = m[current_row][current_row];
00050     if (fabs(pivot) < 1e-12)
00051     {
00052         // Pivot is too small, cannot eliminate
00053         return false;
00054     }
00055
00056     for (int i = current_row + 1; i < total_rows; i++)
00057     {
00058         double factor = m[i][current_row] / pivot;
00059         cout « "Eliminating element in row " « i + 1 « ", column " « current_row + 1 « ":" « endl;
00060         cout « "Multiplying row " « current_row + 1 « " by " « fixed « setprecision(4) « factor
00061              « " and subtracting from row " « i + 1 « "." « endl;
00062         m[i][current_row] = 0.0;
00063         for (int j = current_row + 1; j < total_cols; j++)
00064         {
00065             m[i][j] -= factor * m[current_row][j];
00066         }
00067         cout « endl;
00068     }
00069     return true;
00070 }
00071
00075 int GaussianElimination(vector<vector<double» &m, int rows, int cols)
00076 {
00077     int exchange_count = 0;
00078     int n = min(rows, cols - 1); // Number of variables
00079
00080     for (int k = 0; k < n; k++)
00081     {
00082         cout « "Processing column " « k + 1 « "..." « endl;
00083
00084         // Find the row with the maximum pivot element
00085         int imax = Pivoting(m, k, rows);
00086
00087         // Swap the current row with the pivot row if necessary
00088         if (imax != k)
00089         {
00090             Exchange(m, k, imax);
00091             exchange_count++;
00092         }
00093         else
00094         {
00095             cout « "No need to swap rows for column " « k + 1 « "." « endl;
00096         }
00097
00098         // Check if pivot element is near zero (singular matrix)
```

```
00099         if (fabs(m[k][k]) < 1e-12)
00100         {
00101             cout « "Warning: Pivot element in row " « k + 1 « " is close to zero. The matrix may be
      singular." « endl;
00102             continue; // Skip elimination for this pivot
00103         }
00104 
00105         // Eliminate entries below the pivot
00106         if (!Eliminate(m, k, rows, cols))
00107         {
00108             cout « "Elimination failed for column " « k + 1 « "." « endl;
00109         }
00110 
00111         // Display current matrix state
00112         cout « "Current matrix state:" « endl;
00113         for (int r = 0; r < rows; r++)
00114         {
00115             for (int c = 0; c < cols; c++)
00116             {
00117                 double coeff = round(m[r][c] * 1e12) / 1e12; // Handle floating-point precision
00118                 if (fabs(coeff - round(coeff)) < 1e-12)
00119                 {
00120                     cout « static_cast<long long>(round(coeff)) « "\t";
00121                 }
00122                 else
00123                 {
00124                     cout « fixed « setprecision(2) « coeff « "\t";
00125                 }
00126             }
00127             cout « endl;
00128         }
00129         cout « "-----------------------------------" « endl;
00130     }
00131     return exchange_count;
00132 }
00133 
00137 bool BackSubstitution(const vector<vector<double» &m, int rows, int cols, vector<double> &solution)
00138 {
00139     solution.assign(cols - 1, 0.0);
00140     cout « "Starting back-substitution process..." « endl;
00141     for (int i = rows - 1; i >= 0; i--)
00142     {
00143         // Find the first non-zero coefficient in the row
00144         int pivot_col = -1;
00145         for (int j = 0; j < cols - 1; j++)
00146         {
00147             if (fabs(m[i][j]) > 1e-12)
00148             {
00149                 pivot_col = j;
00150                 break;
00151             }
00152         }
00153 
00154         if (pivot_col == -1)
00155         {
00156             if (fabs(m[i][cols - 1]) > 1e-12)
00157             {
00158                 // Inconsistent equation
00159                 return false;
00160             }
00161             else
00162             {
00163                 // 0 = 0, skip
00164                 continue;
00165             }
00166         }
00167 
00168         double rhs = m[i][cols - 1];
00169         cout « "Calculating x" « pivot_col + 1 « ":" « endl;
00170         for (int j = pivot_col + 1; j < cols - 1; j++)
00171         {
00172             cout « "    " « fixed « setprecision(4) « m[i][j] « " * x" « j + 1
00173                  « " = " « m[i][j] * solution[j] « endl;
00174             rhs -= m[i][j] * solution[j];
00175         }
00176         cout « "    RHS after subtraction = " « rhs « endl;
00177         solution[pivot_col] = rhs / m[i][pivot_col];
00178         cout « "    x" « pivot_col + 1 « " = " « rhs « " / " « m[i][pivot_col]
00179              « " = " « fixed « setprecision(4) « solution[pivot_col] « endl
00180              « endl;
00181     }
00182     return true;
00183 }
00184 
00188 int DetermineRank(const vector<vector<double» &m, int rows, int cols)
00189 {
00190     int rank = 0;
```

```
00191     for (int i = 0; i < rows; i++)
00192     {
00193         bool non_zero = false;
00194         for (int j = 0; j < cols - 1; j++)
00195         {
00196             if (fabs(m[i][j]) > 1e-12)
00197             {
00198                 non_zero = true;
00199                 break;
00200             }
00201         }
00202         if (non_zero)
00203             rank++;
00204     }
00205     return rank;
00206 }
00207
00211 void ShowGeneralSolution(const vector<vector<double» &m, int rows, int cols, int rank)
00212 {
00213     cout « "The system has infinitely many solutions." « endl;
00214     cout « "Solution space dimension: " « (cols - 1 - rank) « endl;
00215
00216     // Identify pivot columns
00217     vector<int> pivots = IdentifyPivots(m, rows, cols);
00218
00219     // Identify free variables
00220     vector<int> free_vars;
00221     for (int j = 0; j < cols - 1; j++)
00222     {
00223         if (find(pivots.begin(), pivots.end(), j) == pivots.end())
00224         {
00225             free_vars.push_back(j);
00226         }
00227     }
00228
00229     // Assign parameters to free variables
00230     int num_free = free_vars.size();
00231     vector<string> params;
00232     for (int i = 0; i < num_free; i++)
00233     {
00234         params.push_back("t" + to_string(i + 1));
00235     }
00236
00237     // Initialize solution vector with parameters
00238     vector<double> particular_solution(cols - 1, 0.0);
00239     vector<vector<double» basis_vectors;
00240
00241     // Find a particular solution by setting all free variables to 0
00242     for (int i = rows - 1; i >= 0; i--)
00243     {
00244         // Find the first non-zero coefficient in the row
00245         int pivot_col = -1;
00246         for (int j = 0; j < cols - 1; j++)
00247         {
00248             if (fabs(m[i][j]) > 1e-12)
00249             {
00250                 pivot_col = j;
00251                 break;
00252             }
00253         }
00254
00255         if (pivot_col == -1)
00256         {
00257             continue; // 0 = 0, skip
00258         }
00259
00260         double rhs = m[i][cols - 1];
00261         for (int j = pivot_col + 1; j < cols - 1; j++)
00262         {
00263             rhs -= m[i][j] * particular_solution[j];
00264         }
00265         particular_solution[pivot_col] = rhs / m[i][pivot_col];
00266     }
00267
00268     // Now, find basis vectors by setting each free variable to 1 and others to 0
00269     for (int i = 0; i < num_free; i++)
00270     {
00271         vector<double> basis(cols - 1, 0.0);
00272         basis[free_vars[i]] = 1.0; // Set the free variable to 1
00273
00274         // Perform back-substitution for pivot variables
00275         for (int r = rank - 1; r >= 0; r--)
00276         {
00277             int pivot_col = pivots[r];
00278             double rhs = 0.0;
00279             for (int j = pivot_col + 1; j < cols - 1; j++)
00280             {
```

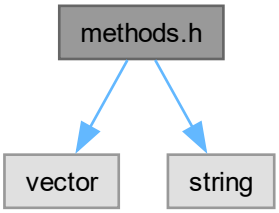
```
00281                 rhs -= m[r][j] * basis[j];
00282             }
00283             basis[pivot_col] = rhs / m[r][pivot_col];
00284         }
00285 
00286         basis_vectors.push_back(basis);
00287     }
00288 
00289     // Display the general solution
00290     cout « "General solution:" « endl;
00291     cout « "x = [";
00292     for (int j = 0; j < cols - 1; j++)
00293     {
00294         cout « fixed « setprecision(4) « particular_solution[j];
00295         if (j < cols - 2)
00296             cout « ", ";
00297     }
00298     cout « "]";
00299 
00300     for (int i = 0; i < num_free; i++)
00301     {
00302         cout « " + " « params[i] « " * [";
00303         for (int j = 0; j < cols - 1; j++)
00304         {
00305             cout « fixed « setprecision(4) « basis_vectors[i][j];
00306             if (j < cols - 2)
00307                 cout « ", ";
00308         }
00309         cout « "]";
00310         if (i < num_free - 1)
00311             cout « " + ";
00312     }
00313     cout « endl
00314          « endl;
00315 }
00316 
00320 vector<int> IdentifyPivots(const vector<vector<double» &m, int rows, int cols)
00321 {
00322     vector<int> pivots;
00323     int n = min(rows, cols - 1);
00324     for (int i = 0; i < n; i++)
00325     {
00326         // Find the pivot column in the current row
00327         int pivot_col = -1;
00328         for (int j = 0; j < cols - 1; j++)
00329         {
00330             if (fabs(m[i][j]) > 1e-12)
00331             {
00332                 pivot_col = j;
00333                 break;
00334             }
00335         }
00336         if (pivot_col != -1)
00337             pivots.push_back(pivot_col);
00338     }
00339     return pivots;
00340 }
```

3.9 methods.h File Reference

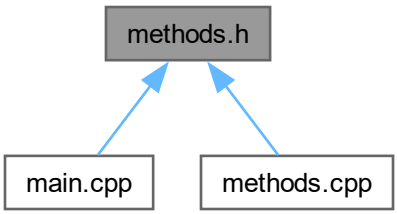
Core computational functions for solving linear systems.

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

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 augmented matrix with partial pivoting.
- int DetermineRank (const std::vector< std::vector< double > > &m, int rows, int cols)

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

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

  Identifies the pivot columns in the matrix.

3.9.1 Detailed Description

Core computational functions for solving linear systems.

Author

Gilbert Young

Date

2024/09/25

This header declares functions for Gaussian elimination with partial pivoting, back-substitution, rank determination, and displaying general solutions.

Definition in file methods.h.

3.9.2 Function Documentation

3.9.2.1 BackSubstitution()

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

Performs back-substitution to find the solution vector.

Parameters

m	The upper triangular matrix after Gaussian elimination.
rows	Number of rows in the matrix.
cols	Number of columns in the matrix (including augmented column).
solution	Reference to store the solution vector.

Returns

true If a unique solution exists.

false If the system is inconsistent.

3.9.2.2 DetermineRank()

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

Determines the rank of the coefficient matrix A (excluding augmented column).

Parameters

m	The augmented matrix [A|b].
rows	Number of rows in the matrix.
cols	Number of columns in the matrix (including augmented column).

Returns

int The rank of the matrix A.

3.9.2.3 GaussianElimination()

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

Performs Gaussian elimination on the augmented matrix with partial pivoting.

Parameters

m	Reference to the augmented matrix [A|b] to be modified.
rows	Number of rows in the matrix.
cols	Number of columns in the matrix (including augmented column).

Returns

int Number of row exchanges performed during elimination.

3.9.2.4 IdentifyPivots()

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

Identifies the pivot columns in the matrix.

Parameters

m	The matrix after Gaussian elimination.
rows	Number of rows in the matrix.
cols	Number of columns in the matrix (including augmented column).

Returns

std::vector<int> A vector containing the indices of the pivot columns.

3.9.2.5 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 (including augmented column).
rank	The rank of the coefficient matrix A.

3.10 methods.h

Go to the documentation of this file.

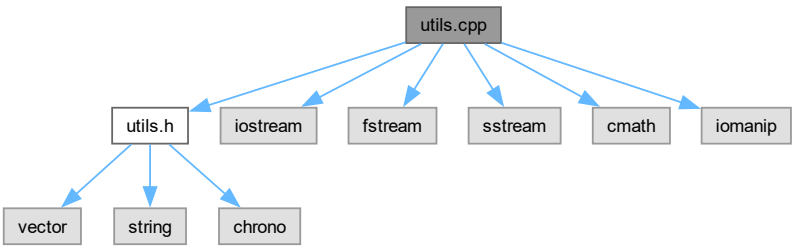
```
00001
00012 #ifndef METHODS_H
00013 #define METHODS_H
00014
00015 #include <vector>
00016 #include <string>
00017
00026 int GaussianElimination(std::vector<std::vector<double» &m, int rows, int cols);
00027
00036 int DetermineRank(const std::vector<std::vector<double» &m, int rows, int cols);
00037
00048 bool BackSubstitution(const std::vector<std::vector<double» &m, int rows, int cols,
      std::vector<double> &solution);
00049
00058 void ShowGeneralSolution(const std::vector<std::vector<double» &m, int rows, int cols, int rank);
00059
00068 std::vector<int> IdentifyPivots(const std::vector<std::vector<double» &m, int rows, int cols);
00069
00070 #endif // METHODS_H
```

3.11 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)

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

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

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

  Displays the unique solution.
- chrono::steady_clock::time_point StartTimer ()
- void StopTimer (const chrono::steady_clock::time_point &start)

3.11.1 Detailed Description

Implementation of utility functions for matrix operations.

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.

Definition in file utils.cpp.

3.11.2 Function Documentation

3.11.2.1 CheckConsistency()

```
bool CheckConsistency (
            const vector< 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.

Definition at line 123 of file utils.cpp.

```
00124 {
00125     for (int i = 0; i < rows; i++)
00126     {
00127         bool all_zero = true;
00128         for (int j = 0; j < cols - 1; j++)
00129         {
00130             if (fabs(m[i][j]) > 1e-12)
00131             {
00132                 all_zero = false;
00133                 break;
00134             }
00135         }
00136         if (all_zero && fabs(m[i][cols - 1]) > 1e-12)
00137         {
00138             return false;
00139         }
00140     }
00141     return true;
00142 }
```

3.11.2.2 DisplaySolution()

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

Displays the unique solution.

Parameters

solution	The solution vector.

Definition at line 147 of file utils.cpp.

```
00148 {
00149     cout « "The system has a unique solution:" « endl;
00150     for (size_t i = 0; i < solution.size(); i++)
00151     {
00152         cout « "x" « i + 1 « " = " « fixed « setprecision(4) « solution[i] « endl;
00153     }
00154 }
```

3.11.2.3 InitMatrix()

```
bool InitMatrix (
            vector< vector< double > > & m,
            const 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.

Definition at line 24 of file utils.cpp.

```
00025 {
00026     ifstream in(filename);
00027     if (!in.is_open())
00028     {
00029         cerr « "Error: Cannot open file " « filename « endl;
00030         return false;
00031     }
00032
00033     // Read the matrix dimensions dynamically
00034     string line;
00035     rows = 0;
00036     cols = 0;
00037     vector<vector<double» temp_matrix;
00038     while (getline(in, line))
00039     {
00040         if (line.empty())
00041             continue; // Skip empty lines
00042         vector<double> row;
```

```
00043         double num;
00044         istringstream iss(line);
00045         while (iss » num)
00046         {
00047             row.push_back(num);
00048         }
00049         if (cols == 0)
00050         {
00051             cols = row.size();
00052         }
00053         else if (static_cast<int>(row.size()) != cols)
00054         {
00055             cerr « "Error: Inconsistent number of columns in the file." « endl;
00056             in.close();
00057             return false;
00058         }
00059         temp_matrix.push_back(row);
00060         rows++;
00061     }
00062     in.close();
00063
00064     if (rows == 0 || cols < 2)
00065     {
00066         cerr « "Error: The matrix must have at least one equation and one variable." « endl;
00067         return false;
00068     }
00069
00070     // Assign to m
00071     m = temp_matrix;
00072     return true;
00073 }
```

3.11.2.4 ShowEquations()

```
void ShowEquations (
            const vector< 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).

Definition at line 78 of file utils.cpp.

```
00079 {
00080     cout « "The current system of linear equations is:" « endl;
00081
00082     for (int i = 0; i < rows; i++)
00083     {
00084         string equation = "";
00085         for (int j = 0; j < cols - 1; j++)
00086         {
00087             // Check if the coefficient is an integer
00088             double coeff = round(m[i][j] * 1e12) / 1e12; // Handle floating-point precision
00089
00090             if (fabs(coeff - round(coeff)) < 1e-12)
00091             {
00092                 equation += to_string(static_cast<long long>(round(coeff))) + " x" + to_string(j + 1);
00093             }
00094             else
00095             {
00096                 // Set precision for floating-point numbers
00097                 equation += to_string(round(m[i][j] * 10000) / 10000.0) + " x" + to_string(j + 1);
00098             }
00099
00100             if (j < cols - 2)
00101                 equation += " + "; // Add space around '+' for better readability
00102         }
00103
00104         // Handle constant term
```

```
00105         double const_term = round(m[i][cols - 1] * 1e12) / 1e12;
00106         if (fabs(const_term - round(const_term)) < 1e-12)
00107         {
00108             equation += " = " + to_string(static_cast<long long>(round(const_term)));
00109         }
00110         else
00111         {
00112             equation += " = " + to_string(round(m[i][cols - 1] * 10000) / 10000.0);
00113         }
00114 
00115         cout « equation « endl; // Output the equation
00116     }
00117     cout « endl; // Add a blank line at the end
00118 }
```

3.11.2.5 StartTimer()

```
chrono::steady_clock::time_point StartTimer ()
```

Definition at line 157 of file utils.cpp.

```
00158 {
00159     return chrono::steady_clock::now();
00160 }
```

3.11.2.6 StopTimer()

```
void StopTimer (
            const chrono::steady_clock::time_point & start)
```

Definition at line 162 of file utils.cpp.

```
00163 {
00164     auto end = chrono::steady_clock::now();
00165     chrono::duration<double> elapsed = end - start;
00166     cout « "Time elapsed: " « elapsed.count() « " seconds." « endl;
00167 }
```

3.12 utils.cpp

Go to the documentation of this file.

```
00001 
00012 #include "utils.h"
00013 #include <iostream>
00014 #include <fstream>
00015 #include <sstream>
00016 #include <cmath>
00017 #include <iomanip>
00018 
00019 using namespace std;
00020 
00024 bool InitMatrix(vector<vector<double» &m, const string &filename, int &rows, int &cols)
00025 {
00026     ifstream in(filename);
00027     if (!in.is_open())
00028     {
00029         cerr « "Error: Cannot open file " « filename « endl;
00030         return false;
00031     }
00032 
00033     // Read the matrix dimensions dynamically
00034     string line;
00035     rows = 0;
00036     cols = 0;
00037     vector<vector<double» temp_matrix;
00038     while (getline(in, line))
00039     {
00040         if (line.empty())
00041             continue; // Skip empty lines
00042         vector<double> row;
00043         double num;
```

```
00044         istringstream iss(line);
00045         while (iss » num)
00046         {
00047             row.push_back(num);
00048         }
00049         if (cols == 0)
00050         {
00051             cols = row.size();
00052         }
00053         else if (static_cast<int>(row.size()) != cols)
00054         {
00055             cerr « "Error: Inconsistent number of columns in the file." « endl;
00056             in.close();
00057             return false;
00058         }
00059         temp_matrix.push_back(row);
00060         rows++;
00061     }
00062     in.close();
00063
00064     if (rows == 0 || cols < 2)
00065     {
00066         cerr « "Error: The matrix must have at least one equation and one variable." « endl;
00067         return false;
00068     }
00069
00070     // Assign to m
00071     m = temp_matrix;
00072     return true;
00073 }
00074
00078 void ShowEquations(const vector<vector<double» &m, int rows, int cols)
00079 {
00080     cout « "The current system of linear equations is:" « endl;
00081
00082     for (int i = 0; i < rows; i++)
00083     {
00084         string equation = "";
00085         for (int j = 0; j < cols - 1; j++)
00086         {
00087             // Check if the coefficient is an integer
00088             double coeff = round(m[i][j] * 1e12) / 1e12; // Handle floating-point precision
00089
00090             if (fabs(coeff - round(coeff)) < 1e-12)
00091             {
00092                 equation += to_string(static_cast<long long>(round(coeff))) + " x" + to_string(j + 1);
00093             }
00094             else
00095             {
00096                 // Set precision for floating-point numbers
00097                 equation += to_string(round(m[i][j] * 10000) / 10000.0) + " x" + to_string(j + 1);
00098             }
00099
00100             if (j < cols - 2)
00101                 equation += " + "; // Add space around '+' for better readability
00102         }
00103
00104         // Handle constant term
00105         double const_term = round(m[i][cols - 1] * 1e12) / 1e12;
00106         if (fabs(const_term - round(const_term)) < 1e-12)
00107         {
00108             equation += " = " + to_string(static_cast<long long>(round(const_term)));
00109         }
00110         else
00111         {
00112             equation += " = " + to_string(round(m[i][cols - 1] * 10000) / 10000.0);
00113         }
00114
00115         cout « equation « endl; // Output the equation
00116     }
00117     cout « endl; // Add a blank line at the end
00118 }
00119
00123 bool CheckConsistency(const vector<vector<double» &m, int rows, int cols)
00124 {
00125     for (int i = 0; i < rows; i++)
00126     {
00127         bool all_zero = true;
00128         for (int j = 0; j < cols - 1; j++)
00129         {
00130             if (fabs(m[i][j]) > 1e-12)
00131             {
00132                 all_zero = false;
00133                 break;
00134             }
00135         }
00136         if (all_zero && fabs(m[i][cols - 1]) > 1e-12)
```

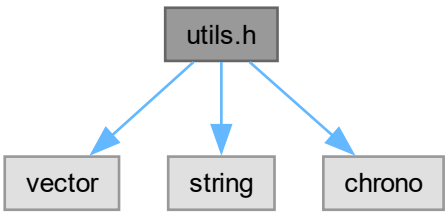
```
00137         {
00138             return false;
00139         }
00140     }
00141     return true;
00142 }
00143
00147 void DisplaySolution(const vector<double> &solution)
00148 {
00149     cout « "The system has a unique solution:" « endl;
00150     for (size_t i = 0; i < solution.size(); i++)
00151     {
00152         cout « "x" « i + 1 « " = " « fixed « setprecision(4) « solution[i] « endl;
00153     }
00154 }
00155
00156 // Timing functions
00157 chrono::steady_clock::time_point StartTimer()
00158 {
00159     return chrono::steady_clock::now();
00160 }
00161
00162 void StopTimer(const chrono::steady_clock::time_point &start)
00163 {
00164     auto end = chrono::steady_clock::now();
00165     chrono::duration<double> elapsed = end - start;
00166     cout « "Time elapsed: " « elapsed.count() « " seconds." « endl;
00167 }
```

3.13 utils.h File Reference

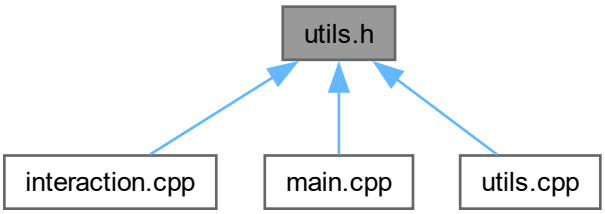
Utility functions for matrix initialization and display.

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

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.
- std::chrono::steady_clock::time_point StartTimer ()
- void StopTimer (const std::chrono::steady_clock::time_point &start)

3.13.1 Detailed Description

Utility functions for matrix initialization and display.

Author

Gilbert Young

Date

2024/09/25

Definition in file utils.h.

3.13.2 Function Documentation

3.13.2.1 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.

3.13.2.2 DisplaySolution()

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

Displays the unique solution.

Parameters

solution	The solution vector.

3.13.2.3 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.

3.13.2.4 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).

3.13.2.5 StartTimer()

```
std::chrono::steady_clock::time_point StartTimer ()
```

Definition at line 157 of file utils.cpp.

```
00158 {
00159     return chrono::steady_clock::now();
00160 }
```

3.13.2.6 StopTimer()

```
void StopTimer (
            const std::chrono::steady_clock::time_point & start)
```

3.14 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 #include <chrono>
00014
00025 bool InitMatrix(std::vector<std::vector<double» &m, const std::string &filename, int &rows, int
      &cols);
00026
00034 void ShowEquations(const std::vector<std::vector<double» &m, int rows, int cols);
00035
00045 bool CheckConsistency(const std::vector<std::vector<double» &m, int rows, int cols);
00046
00052 void DisplaySolution(const std::vector<double> &solution);
00053
00054 // Timing functions
00055 std::chrono::steady_clock::time_point StartTimer();
00056 void StopTimer(const std::chrono::steady_clock::time_point &start);
00057
00058 #endif // UTILS_H
```

Index