Linear Algebra Library Documentation

Header File ($\boldsymbol{MatrixOperation.h}$)

The header file includes the following function definitions declared in the header file along with the header file of "$\boldsymbol{vector.h}$", and "$\boldsymbol{iostream.h}$" :

- $\#\boldsymbol{include} < \boldsymbol{vector} >$: This is used to initialize vector for creation of dynamic array
- $\#\boldsymbol{include} < \boldsymbol{iomanip} >$: This is used for changing the printing format
- $\boldsymbol{defineMatrix}$: Using this function, the size of rows and columns of the matrices is defined using $\boldsymbol{resize}()$ function. This can be understood by following example:
    - Expression for $defineMatrix()$ can be given as $defineMatrix(int\ row, int\ col)$. When called produces the following result:
        - mat = {{1,2,3},{4,5,6},{7,8,9},{1,5,9}};
        - This means there are 4 rows and 3 columns
        - Which result in our matrix to be defined as $defineMatrix(int\ 4, int\ 3)$
        - Now, this will create an empty matrix of order $(4 * 3)$ which looks like: $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
- $\boldsymbol{fillMatrix}$: This function is used to store all the values entered by the user into their respective positions defined in the matrix. This can be done in two steps:
    - In the first step, it will assign all the index positions in the matrix defined with the reference of the matrix entered by the user. Also, it will check if all the column values are same in each row defined and if not, it will give an error. So, in the first step, the matrix entered by the user looks like:
        - $fillMatrix(matrix) = \begin{bmatrix} mat(0,0) & mat(0,1) & mat(0,2) \\ mat(1,0) & mat(1,1) & mat(1,2) \\ mat(2,0) & mat(2,1) & mat(2,2) \\ mat(3,0) & mat(3,1) & mat(3,2) \end{bmatrix}$
    - In the second step, the matrix will start allocating the values entered by the user to the respective index values. In this step, the $fillMatrix$ will look like:
        - $fillMatrix(matrix) = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 1 & 5 & 9 \end{bmatrix}$
- $\boldsymbol{operator} *$ **:** In this function, the logic for matrix multiplication is implemented. For this:
    - Firstly, check the order of the matrices to be multiplied. For that, the column size of the first matrix must be equal to the row size of the second matrix, e.g. $(\boldsymbol{4} * \boldsymbol{2}) * (\boldsymbol{2} * \boldsymbol{3})$. Else an error will state that the size of the matrices is incompatible. This can be seen with the help of an example:

- $matrix1(int\ rowMat1, colMat1) = \begin{bmatrix}1 & 2 & 3 & 4\\5 & 6 & 7 & 8\end{bmatrix}$;

  $matrix2(int\ rowMat2, colMat2) = \begin{bmatrix}1 & 2\\3 & 4\\5 & 6\\7 & 8\end{bmatrix}$;
- Here, $rowMat1 = 2, colMat1 = 4, rowMat2 = 4, colMat2 = 2$. The first step will if $colMat1 == rowMat2$. If the condition states true, then only multiplication will be performed on both the matrices, else it will not

o Once that checked, now the loop will apply for order decided on the final matrix vector after multiplication, i.e., $\boldsymbol{matRes(rowMat1, colMat2)}$. This loop is ran for the size of the column for the first matrix such that the $\boldsymbol{matRes(rowMat1, colMat2)} += (\boldsymbol{mat1(rowMat1, colMat1)} * \boldsymbol{mat2(colMat1, colMat2)})$. Now, considering the above example, store the product matrix after multiplication:

- Since, the dimension of the column of matrix 1 is equal to the row dimension of matrix 2. Therefore, on performing multiplication, equation can be written as: $matRes(2,2) = matRes(2,2) + \big(matrix1(2,4) * matrix(4,2)\big)$;
- Also, $matRes(rowMat1, colMat2)$ matrix is initially a zero matrix and after the multiplication it is updated
- $matRes(2,2) = \begin{bmatrix}0 & 0\\0 & 0\end{bmatrix} + \left(\begin{bmatrix}1 & 2 & 3 & 4\\5 & 6 & 7 & 8\end{bmatrix} * \begin{bmatrix}1 & 2\\3 & 4\\5 & 6\\7 & 8\end{bmatrix}\right) = \begin{bmatrix}49 & 60\\114 & 140\end{bmatrix}$

- $\boldsymbol{matTrans}$ **:** With the help of this function, transposition of a matrix can be done using a loop logic of interchanging the rows and columns of the matrix entered by the user. It can be done as:
  - Consider a matrix, $matrix(row, col) = \begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\\1 & 5 & 9\end{bmatrix}$
  - Now, for transpose of this matrix just interchange the row elements with column and vice-versa, this can be written as:
    - $matTrans(col, row) = \begin{bmatrix}1 & 4 & 7 & 1\\2 & 5 & 8 & 5\\3 & 6 & 9 & 9\end{bmatrix}$
- $\boldsymbol{printMatrix}$: The matrix output can be displayed using this function. This is done using a double reference variable in which the $rvalue$ is compared with the input. When calculating the transpose, only one reference variable would work so, the matrix is just copied to the reference variable to equate the $lvalue$ and return the matrix to display as the output. The $\boldsymbol{Precisionset(int\ n)}$ function is used for getting the precise value with $'\boldsymbol{n}'$ being the number of decimal places the output will be

Main File ($\boldsymbol{MainFunction.cpp}$)

The main file is just being used for taking input from the user and display the output that user desire. It includes the header file such as "$\boldsymbol{MatrixOperations.h}$", "$\boldsymbol{vector.h}$", and "$\boldsymbol{iostream.h}$":

- Firstly, using the $\boldsymbol{fillMatrix}()$ function, the first and second matrices input is taken from the user using a format of $(\{\{\}, \{\}, \{\}, \ldots\})$

- Based on the input taken, the $\boldsymbol{fillMatrix}()$ function will take care of calculating the size and order of the matrices entered
- Now, just multiply the matrices to get the matrix multiplication code working, i.e., $\boldsymbol{mat3} = (\boldsymbol{mat1} * \boldsymbol{mat2})$
- The display of the output of $\boldsymbol{mat3}$ can be done by writing $\boldsymbol{printMatrix}(\boldsymbol{mat3})$
- Alternatively, the transpose of $\boldsymbol{mat3}$ can also be displayed as $printMatrix(matTrans(mat3))$
- In addition to that, transpose of a matrix entered by the user can also be calculated and printed using a formula: $\boldsymbol{printMatrix}(\boldsymbol{matTrans}(\boldsymbol{mat4}))$, where, the format of the matrix is same as the format used for the input of the first two matrices

Compiling the Code

- There are two $.cpp$ format files named as $Functions.cpp$ and $MainFunction.cpp$
- The $Functions.cpp$ file represents all the functions being used for defining the matrices and performing operations on it
- The $MainFunction.cpp$ file represents the $main()$ function from where the functions being called to get the output

For Windows

- Using the Visual Studio Command Prompt, type the following command:
  $\boldsymbol{cl\ /EHsc\ MainFunction.cpp\ Functions.cpp}$
- By typing the executable file $\boldsymbol{main.exe}$ in command window, the resulting output can be seen

For Linux

- In the terminal, type the following command:
  $$\boldsymbol{g++\ MainFunction.cpp\ Functions.cpp\ --std=c++11\ -o\ Main.out}$$
- By typing the executable file $\boldsymbol{./Main.out}$ in the terminal, the resulting output can be seen

Supporting Examples

Example 1:

Code:

```
#include <vector>
#include <iomanip>


/// This will define the size of matrix ///
std::vector<std::vector<double>> defineMatrix(int dimM, int dimN);
/// This will fill the matrix entered by the user ///
```

```
std::vector<std::vector<double>> fillMatrix(const
std::initializer_list<std::initializer_list<double>> &mat);

/// This will perform the matrix multiplication logic of two matrices ///

std::vector<std::vector<double>> operator*(std::vector<std::vector<double>>
&mat1, std::vector<std::vector<double>> &mat2);

/// This will perform Matrix Transposition on the matrix entered by the
user///

std::vector<std::vector<double>> matTrans(std::vector<std::vector<double>>
&matrix);

/// This will display the matrix using reference variable of two matrices ///

void printMatrix(std::vector<std::vector<double>> &&mat);

/// This will equate the rvalue and lvalue of the matrix and return the
result to the above print_matrix function ///

void printMatrix(std::vector<std::vector<double>> &mat);

/// Define Matrix will generate a empty matrix of the size mentioned in the
input of the matrices ///

std::vector<std::vector<double>>defineMatrix(int row, int col)

{

      std::vector<std::vector<double>>matrix(row);        /// This will
resize the row: (matrix.resize(row);)///

      for (int i = 0;i < row;i++)                         /// This will
resize the column for each row ///

            matrix[i].resize(col);

      return matrix;

}


/// Fill Matrix will initialize a list which will start filling the values
with respect to the matrices entered ///

std::vector<std::vector<double>> fillMatrix(const
std::initializer_list<std::initializer_list<double>> &mat)

{

    size_t n_rows = mat.size(), n_cols = mat.begin()->size();   ///
Initialize the size of the rows and columns ///

    for(unsigned int i = 0; i<n_rows; ++i)                    /// Defining
the size of the rows for the matrix entered///

    {
```

```
        if((mat.begin()+i)->size()!=n_cols)                    /// Check for
the columns that each row contains the same no. of elements ///
        {
            std::cout << "Error: Object initialization failed. Number of
elements in each row must be same." << std::endl;
            std::vector<std::vector<double>> matrix(n_rows);    /// Resizing
an empty matrix if not true ///
            for(unsigned int i = 0; i<n_rows; ++i)
            {
                matrix[i].resize(n_cols);
            }
            break;
        }
    }

    std::vector<std::vector<double>> matrix(n_rows);            /// Creation
of a vector for storing the values entered by the user ///
    for(unsigned int i = 0; i<n_rows; ++i)
    {
        matrix[i].resize(n_cols);
        for(unsigned int j = 0; j<n_cols; ++j)
        {
            matrix[i][j] = *((mat.begin()+i)->begin()+j);       /// This will
store the row and column value to their respective positions ///
        }
    }

    return matrix;
}

/// This will perform matrix multiplication on two matrices and store in
another matrix ///
std::vector<std::vector<double>> operator*(std::vector<std::vector<double>>
&mat1, std::vector<std::vector<double>> &mat2)
{
```

```
    /// Initializing the row and column values for the matrices///
      int rowMat1 = mat1.size();
      int colMat1 = mat1[0].size();
      int rowMat2 = mat2.size();
      int colMat2 = mat2[0].size();

      std::vector<std::vector<double>>matRes = defineMatrix(rowMat1,
colMat2);    /// Defining the matrix for resulting matrix ///
      if (colMat1 == rowMat2)
/// The order for column of Matrix1 should be equal to the row of Matrix2 ///
      {
            for (int i = 0; i < rowMat1; i++)
/// Defining the row of the product matrix ///
                  for (int j = 0; j < colMat2; j++)
/// Defining the column of the product matrix ///
                        for (int k = 0; k < colMat1; k++)
/// Check for the common order of the user entered matrices ///
                              matRes[i][j] = matRes[i][j] + (mat1[i][k] *
mat2[k][j]);    /// Logic for the multiplication matrix ///
      }
      else
        std::cout<<"The row and column of matrices are not same"<<std::endl;
      return matRes;
}

/// This will perform the matrix transposition entered by the user ///
std::vector<std::vector<double>>matTrans(std::vector<std::vector<double>>
&matrix)
{
    size_t r = matrix.size();                                    ///
Getting the row of the matrix entered ///
    size_t c = matrix[0].size();                                 ///
Getting the column of the matrix entered ///

    std::vector<std::vector<double>> Trans = defineMatrix(c,r);     ///
Defining the matrix size of the transposed matrix ///
```

```
    for (unsigned int i = 0; i < r; ++i)
            for (unsigned int j = 0; j < c; ++j)
            {
                  Trans[j][i] = matrix[i][j];                              ///
Logic for changing rows into columns and vice-versa ///
            }
    return Trans;
}

/// This will display the final matrix ///
void printMatrix(std::vector<std::vector<double>> &&mat)
{
    size_t n_rows = mat.size(), n_cols = mat[0].size();
    std::cout << std::scientific << std::setprecision(4);          /// This
will set the precision value of the result up to 4 decimal places ///
    for(unsigned int i = 0; i<n_rows; ++i)
    {
        for(unsigned int j = 0; j<n_cols; ++j)
        {
            std::cout<<mat[i][j]<<" ";
        }
        std::cout<<std::endl;
    }
}

/// This will equate the lvalue with rvalue of the matrix value using
reference variable ///
void printMatrix(std::vector<std::vector<double>> &mat)
{
    return printMatrix(std::move(mat));
}
```

Main File:

```
#include "MatrixOperations.h"

#include <iostream>

#include <vector>


int main()

{

    /// Enter the First Matrix here in the format: ({{},{},{},...}) ///

    std::vector<std::vector<double>> mat1 =
fillMatrix({{1,2,3,4},{5,6,7,8},{1,3,5,7}});

    /// Enter the Second Matrix here in the format: ({{},{},{},...}) ///

    std::vector<std::vector<double>> mat2 =
fillMatrix({{9,8,7},{6,5,4},{3,2,1},{0,5,1}});

    /// Matrix Multiplication of the Multidimensional Matrices ///

    std::vector<std::vector<double>> mat3 = mat1*mat2;

    /// Either print the Resulting Matrix after Multiplication ///

    std::cout<<"Matrix Multiplication: "<<std::endl;

    printMatrix(mat3);

   std::vector<std::vector<double>> mat4 =
fillMatrix({{1,2,3,4,5},{6,7,8,9,0}});

    /// Transposition of a matrix can be done by using the following format:
mat4 = ({{},{},{},...}) ///

    std::cout<<std::endl<<"Transpose Matrix: "<<std::endl;

    ///printMatrix(matTrans(mat3));

    printMatrix(matTrans(mat4));

}
```

Input:

- Input given by user for the matrices:
  - $mat1 = fillMatrix(\{\{1,2,3,4\},\{5,6,7,8\},\{1,3,5,7\}\});$
  - $mat2 = fillMatrix(\{\{9,8,7\},\{6,5,4\},\{3,2,1\},\{0,5,1\}\});$
  - $mat4 = fillMatrix(\{\{1,2,3,4,5\},\{6,7,8,9,0\}\});$

- To run the code (for Windows), just type $cl\ /EHsc\ MainFunction.cpp\ Function.cpp$, this will create an executable file $MainFunction.exe$. And then type MainFunction.exe to see the final output.

Output:

Matrix Multiplication:

$$\begin{bmatrix} 3.0000e+001 & 4.4000e+001 & 2.2000e+001 \\ 1.0200e+002 & 1.2400e+002 & 7.4000e+001 \\ 4.2000e+001 & 6.8000e+001 & 3.1000e+001 \end{bmatrix}$$

Transpose Matrix:

$$\begin{bmatrix} 1.0000e+000 & 6.0000e+000 \\ 2.0000e+000 & 7.0000e+000 \\ 3.0000e+000 & 8.0000e+000 \\ 4.0000e+000 & 9.0000e+000 \\ 5.0000e+000 & 0.0000e+000 \end{bmatrix}$$

Example 2:

Code:

The code for all the matrices will be same and only the main function will be changed where the input is taken from the user.

Main File:

```
#include "MatrixOperations.h"

#include <iostream>

#include <vector>


int main()

{

    /// Enter the First Matrix here in the format: ({{},{},{},...}) ///

    std::vector<std::vector<double>> mat1 =
fillMatrix({{1,2,3,4},{5,6,7,8},{1,3,5,7}});

    /// Enter the Second Matrix here in the format: ({{},{},{},...}) ///

    std::vector<std::vector<double>> mat2 =
fillMatrix({{9,8,7},{6,5,4},{3,2,1},{0,5,1}});

    /// Matrix Multiplication of the Multidimensional Matrices ///

    std::vector<std::vector<double>> mat3 = mat1*mat2;

    /// Either print the Resulting Matrix after Multiplication ///
```

```
    std::cout<<"Matrix Multiplication: "<<std::endl;

    printMatrix(mat3);

    /// Transposition of a matrix can be done by using the following format:
mat4 = ({{},{},{},...}) ///

    std::cout<<std::endl<<"Transpose Matrix: "<<std::endl;

    printMatrix(matTrans(mat3));

    ///printMatrix(matTrans(mat4));

}
```

Input:

- Input given by user for the matrices:
  - $mat1 = fillMatrix(\{\{1,2,3,4\},\{5,6,7,8\},\{1,3,5,7\}\});$
  - $mat2 = fillMatrix(\{\{9,8,7\},\{6,5,4\},\{3,2,1\},\{0,5,1\}\});$
- For compiling, follow the same procedure as mentioned in ***Example* 1**

Output:

Matrix Multiplication:

$$\begin{bmatrix} 3.0000e+001 & 4.4000e+001 & 2.2000e+001 \\ 1.0200e+002 & 1.2400e+002 & 7.4000e+001 \\ 4.2000e+001 & 6.8000e+001 & 3.1000e+001 \end{bmatrix}$$

Transpose Matrix:

$$\begin{bmatrix} 3.0000e+001 & 1.0200e+002 & 4.2000e+001 \\ 4.4000e+001 & 1.2400e+002 & 6.8000e+001 \\ 2.2000e+001 & 7.4000e+001 & 3.1000e+001 \end{bmatrix}$$