Introduction

The project is about building a library for matrix. The language we have chose to implement this project is C++. We use vector of vectors to implement the matrix. The interface of the project is made using classes in C++.

Concepts Used :

Operator overloading
Generic programming
Friend functions
Templates
Class

Implementation Details :

Since C++ by default doesn't support matrix operations by default we tried to implement the library for the same. C++ by default doesn't support +,-,* operations for matrix. So we use the concept of operator overloading to overload the needed operators.

Also some of the operators cannot be overloaded since the definitions are not defined in the libraries we use friend function to implement some of those.

Using member function of the matrix class we have developed determinant inverse functions for each matrix.

Functionalities implemented :

- Addition of matrices - Overloaded + operator
- Subtraction of matrices - Overloaded - operator
- Multiplication of matrices - Overloaded * operator
- Scalar Multiplication - Overloading * (Both friend and member function)
- Copy Assignment operator
- Equality of Matrices - Overloaded == operator
- Indexing - []
- AppendRow - It takes in the matrix and appends the row with default value or the value mentioned.
- AppendCol - It takes in the matrix and appends the column with default value or the value mentioned.
- Rows - Returns the number of rows of the matrix
- Cols - Returns the number of columns of the matrix
- Cout - Overloaded << operator to print the matrix
- Det - returns the determinant of the matrix
- Inverse - Finds inverse of a square matrix
- Solving linear equation

Interface :

```
template<class T>
class matrix
{
   private:
      template <typename TE> friend matrix<TE> operator*(int i,const
matrix<TE>& rhs);
      template <typename TE> friend matrix<TE> operator*(const
matrix<TE>& a, const matrix<TE>& b);
      template <typename TE> friend ostream& operator<<(ostream
&os, const matrix<TE>& x);
      vector<vector<T>> mat;
      int N,M;

   public:
      matrix<T>(): N(0), M(0) {}
      matrix<T>(int N,int M,T init=0);

      matrix<T>(const std::vector<std::vector<T>> &m);

      //float det();
      unsigned Rows() const;
      unsigned Cols() const;
      matrix<T>& AppendRow(T init=0);
      matrix<T>& AppendCol(T init=0);
      bool operator==(const matrix<T>& rhs);
      matrix<T> operator*(int i);
      matrix<T>& operator=(const matrix<T>& rhs);
      matrix<T> operator+(const matrix<T>& rhs);
```

```
        matrix<T> operator-(const matrix<T>& rhs);
        std::vector<T>& operator[](unsigned i) { return mat[i]; }
        std::vector<T> operator[](unsigned i) const { return mat[i]; }
        matrix<T> Coff(unsigned i, unsigned j) const;
        static T Det(const matrix<T>& x);
        T Det() const;
        matrix<T> Inv() const;
        matrix<T> LinSolve(const matrix &A, const matrix &b);

};
```

Conclusion :

Matrix library with operations like +,-,*,inverse , determinent and some more functionalities are achieved.

Whenever we use the matrix this library can be used for basic operations of matrices in C++.

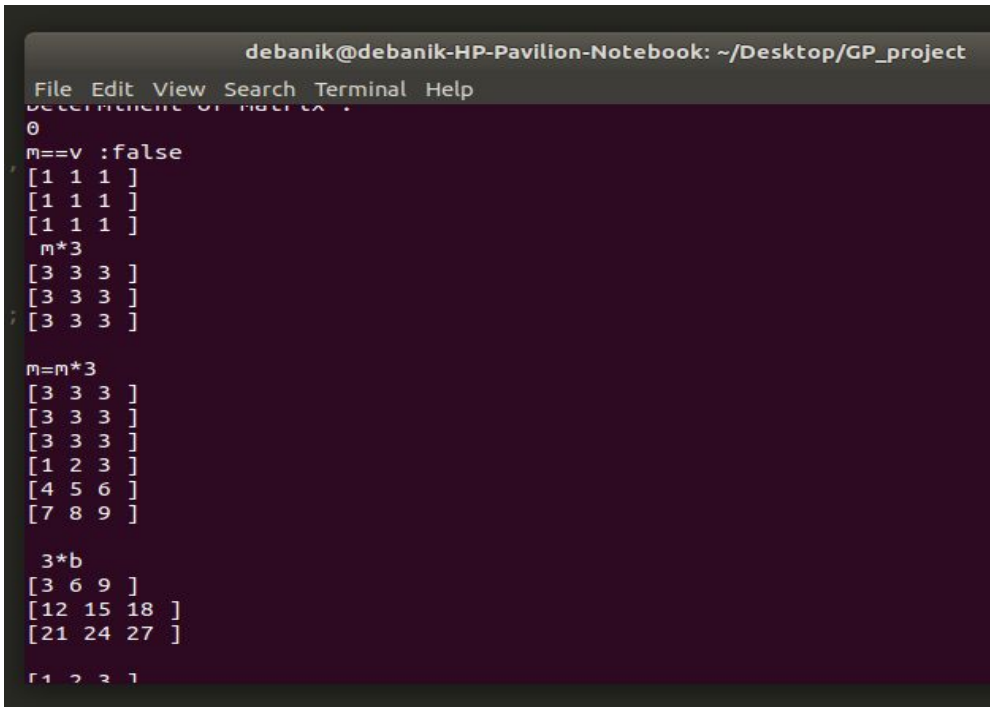
Improvements possible :

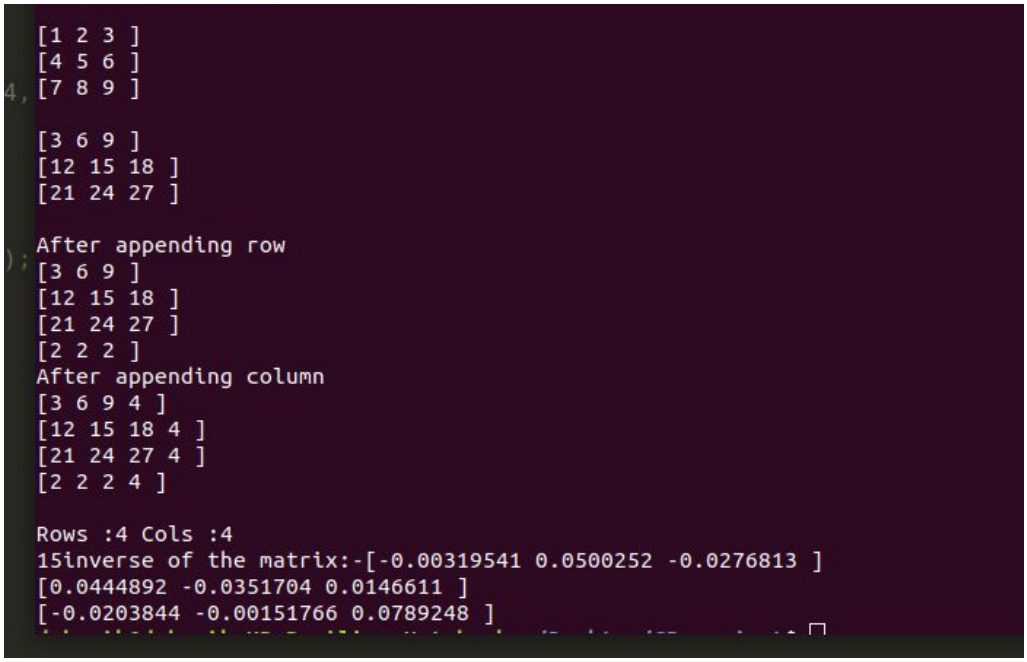
More general algorithm for inverse and determinent that works for matrix of all dimensions can be developed.

Support for more operators like *=, += can be implemented.

Iterators for the matrix could be developed.

Results


debanik@debanik-HP-Pavilion-Notebook: ~/Desktop/GP_project
File Edit View Search Terminal Help
0
m==v :false
[1 1 1]
[1 1 1]
[1 1 1]
m*3
[3 3 3]
[3 3 3]
[3 3 3]
m=m*3
[3 3 3]
[3 3 3]
[3 3 3]
[1 2 3]
[4 5 6]
[7 8 9]
3*b
[3 6 9]
[12 15 18]
[21 24 27]


[1 2 3]
[4 5 6]
[7 8 9]
[3 6 9]
[12 15 18]
[21 24 27]
After appending row
[3 6 9]
[12 15 18]
[21 24 27]
[2 2 2]
After appending column
[3 6 9 4]
[12 15 18 4]
[21 24 27 4]
[2 2 2 4]
Rows :4 Cols :4
15inverse of the matrix:-[-0.00319541 0.0500252 -0.0276813]
[0.0444892 -0.0351704 0.0146611]
[-0.0203844 -0.00151766 0.0789248]