

CS205 C/C++ Projcet - Matrix

Name: 李伯岩(Boyan Li), 邓淞航(Songhang Deng), 彭书博(Shubo Peng)

SID: 11912914, 11912918, 12010905

Develop Environment:

Windows10, MinGw64, C++ 14, g++ (GCC) 7.4.0, Visual Studio 2019, openCV 4.4.0

Part 1 - Analysis

The purpose of this project is to build a powerful and easy to use matrix library based on C/C++ language. The overall structure of the project is as follows:

File_1 : <Structure.hpp>

In <Stucture.hpp>, we defined all the data structure needed.

Class	Relation	Role
AbstractTensor<Type>	Null	Base Tensor Class
Matrix<Type>	Inherit AbstractTensor<Type>	Implemented Matrix Data Structure Class, supply all the matrix methods. Matrix dimension is 2 valid.
HessenbergHouseholderUtils<inner>	Inner class in Matrix<Type>	Integrate the HouseHolder Algorithm to calculate matrix eigenvalues.
Tensor<Type>	Inherit AbstractTensor<Type>	Implemented Tensor Data Structure Class, supply all the tensor methods. Tensor dimension is arbitrary valid.
SolveExpressions	Null	Provide real-time interaction between the system and users.

File_2 : <Exceptions.hpp>

In <Exceptions.hpp>, we defined all the possible exceptions during running.

Class	Inheritance	Throw Condition
BaseException	Null	Unknown Exception.
CrossProductVectorNotValidException	BaseException	When two vector cannot do cross-product, such as one is not a vector.
DefinedConjugationFunctionNotValidException	BaseException	When the user-defined conjugation function is not valid.
DeterminantMatrixNotSquareException	BaseException	When a un-square matrix to calculate determinant.
DotMatrixShapeNotValid	BaseException	When two matrix do dot with different shape.
MatrixConvolutionNotValidException	BaseException	When matrix and kernel do convolution, but shape and parameters are not valid.
MatrixEquationNoSolutionException	BaseException	When a matrix equation has no solution.
MatrixEquationParameterException	BaseException	When a matrix equation parameter is not valid.
MatrixIndexNotValidException	BaseException	When matrix index is not valid.
MatrixInverseNotValidException	BaseException	When a matrix has no inverse.
MatrixMultiplicationShapeNotValidException	BaseException	When two matrix do multiplication, but shape is not valid.
MatrixNotDenseException	BaseException	When a operation need a dense matrix, but input is not.
MatrixNotSparseException	BaseException	When a operation need a sparse matrix, but input is not.
MatrixReshapeNotValidException	BaseException	When the shape of reshape is not proper to matrix now.

Class	Inheritance	Throw Condition
MatrixShapeNotSameException	BaseException	When two matrix has different shape.
MatrixShapeNotValidException	BaseException	When the matrix shape is not valid for some operations.
MatrixTraceNotSquareException	BaseException	The calculate the trace, but the matrix is not square.

Part 2 - API Document

Part2.1 Class <AbstractTensor>

No implemented method.

Part2.2 Class <Matrix>

Member methods

Name	Privilege	Explanation
Matrix()	public	dense matrix default constructor
Matrix(int rows, int cols)	public	dense matrix constructor with initial rows and cols
Matrix(int rows, int cols, int val)	public	dense matrix constructor with initial rows, cols and values
Matrix(const vector<vector>& vec)	public	dense matrix constructor with a vector
Matrix(map<pair<int, int>, Type>& data_sparse)	public	sparse matrix constructor with initial map format data
Matrix(const Matrix& matrix)	public	overload copy-constructor
Matrix& operator=(const Matrix& matrix)	public	overload assignment operator
~Matrix()	public	default destructor
air<int, int> shape() const	public	return matrix shape
int size() const	public	return the number of elements in matrix
int cols() const	public	return the number of columns
int rows() const	public	return the number of rows
bool is_square() const	public	return the matrix is whether square
bool is_vector() const	public	return the matrix is whether a vector
Matrix reshape(int rows, int cols, bool inplace=false)	public	reshape the matrix to specific (rows, cols), with inplace=true, it will change itself
Matrix operator+(const Matrix& matrix)	public	matrix + matrix (same shape)
Matrix operator+(const Type& val)	public	matrix + const value
Matrix operator-(const Matrix& matrix)	public	matrix - matrix (same shape)
Matrix operator-(const Type& val)	public	matrix - const value
Matrix operator*(const Matrix& matrix)	public	matrix * matrix (valid shape)
Matrix operator*(const Type& val)	public	matrix * const value
Matrix operator/(const Type& val)	public	matrix / const value

Name	Privilege	Explanation
Matrix transposition(bool inplace=false)	public	transpose the matrix, if inplace=true, change itself
Matrix conjugation(bool inplace=false)	public	conjugate the matrix with default complex conjugation, if inplace=true, change itself
Matrix conjugation(Type (*pf)(const Type&), bool inplace=false)	public	conjugate the matrix with user-defined conjugation function (*pf), if inplace=true, change itself
Matrix dot(const Matrix& matrix)	public	dot operation with another matrix
Matrix mul(const Matrix& matrix)	public	multiplication with another matrix
Matrix cross_product(const Matrix& matrix)	public	calculate cross-product with another vector
Type determinant() const	public	calculate the determinant
Matrix inverse(bool inplace=false)	public	calculate the inverse of matrix (if exists)
Type row_max(int row) const	public	return the max value in specific row
Type row_min(int row) const	public	return the min value in specific row
Type row_sum(int row) const	public	return the sum of specific row
Type row_avg(int row) const	public	return the average value of specific row
Type col_max(int col) const	public	return the max value in specific column
Type col_min(int col) const	public	return the min value in specific column
Type col_sum(int col) const	public	return the sum of specific column
Type col_avg(int col) const	public	return the average value of specific column
Matrix rows_max() const	public	return the max values in every row
Matrix cols_max() const	public	return the max values in every column
Type matrix_max() const	public	return the max value of the matrix
Matrix rows_min() const	public	return the min value of every row
Matrix cols_min() const	public	return the min value of every column
Type matrix_min() const	public	return the min value of the matrix

Name	Privilege	Explanation
Matrix rows_sum() const	public	return the sum of every row
Matrix cols_sum() const	public	return the sum of every column
Type matrix_sum() const	public	return the sum of the matrix
Matrix rows_avg() const	public	return the average value of every row
Matrix cols_avg() const	public	return the average value of every column
Type matrix_avg() const	public	return the average value of the matrix
Type trace() const	public	return the trace of the square matrix
Matrix slice(pair<int, int> rows, pair<int, int> cols, int rows_step=1, int cols_step=1) const	public	silce the matrix, including the row range, column range and row/column step
int rank() const	public	return the rank of the matrix
string getMatrixType() const	public	return the name of matrix type (dense or sparse)
Matrix convolution(const Matrix& kernel, int stride=1, int padding=0)	public	return the convolution result, using the kernel, stride (default=1), padding (default=0) as parameters
void copyToMatrix(Matrix& matrix) const	public	copy to the matrix
vector getEigenValues() const	public	get the eigenvalues of the matrix
Matrix convertToHessenbergMatrix() const	public	convert to hessenberg matrix
Matrix getEigenVectors() const	public	get the eigenvectors of the matrix
Type norm(int order = 2)	public	return the norm-1 or norm-2 of the matrix

Member variables

Name	Privilege	Explanation
vector<vector> data	private	store the data for dense matrix
map<pair<int,int>,Type> data_sparse	private	store the data for sparse matrix
int matrixType	private	the matrix is whether dense or sparse, 0 means dense and 1 means sparse

Static methods

Name	Privilege	Explanation
Matrix zeros(int rows, int cols)	public	return a all-zero matrix
Matrix ones(int rows, int cols)	public	return a all-one matrix
Matrix values(int rows, int cols, Type val)	public	return a all-value matrix
Matrix eyes(int n, int val=1)	public	return a diagonal matrix with val (default 1)
Mat MatrixToMat_Float(const Matrix& matrix)	public	convert a float matrix to cv::Mat
Mat MatrixToMat_Double(const Matrix& matrix)	public	convert a double matrix to cv::Mat
Mat MatrixToMat_Int(const Matrix& matrix)	public	convert a int matrix to cv::Mat
Mat MatrixToMat_Uchar(const Matrix& matrix)	public	convert a uchar matrix to cv::Mat
Matrix MatToMatrix_Float(const Mat& mat)	public	convert a cv::Matto float matrix
Matrix MatToMatrix_Double(const Mat& mat)	public	convert a cv::Matto double matrix
Matrix MatToMatrix_Int(const Mat& mat)	public	convert a cv::Matto intmatrix
Matrix MatToMatrix_Uchar(const Mat& mat)	public	convert a cv::Mat to uchar matrix
Matrix solveLinearEquations(const Matrix& A, const Matrix& b)	public	return the matrix equation solution x of $Ax=b$
Matrix Guess(const Matrix& matrix)	public	return matrix after apply Guess method
Matrix Guess(const Matrix& matrix, int*& outOrder)	public	return matrix after apply Guess method, with changing the order of rows
Matrix DenseToSparse(Matrix& dense_matrix, bool inplace = false)	public	convert dense matrix to sparse matrix, if inplace=true, change itself
Matrix SparseToDense(Matrix& sparse_matrix, bool inplace = false)	public	convert sparse matrix to dense matrix, if inplace=true, change itself
Matrix getRForAStep(Matrix R, int i)	private	get matrix R druing Householder algorithm
bool isConvergence(Matrix& matrix)	public	check the matrix is whether convegence during Householder algorithm

Part 3 - Result & Verification

Note : The result verification display will test according to the project documentation.

Part 3.1

1) It supports all matrix sizes, from small fixed-size matrices to arbitrarily large dense matrices, and even sparse matrices (Add: try to use efficient ways to store the sparse matrices). (10 points)↵

Test Code:

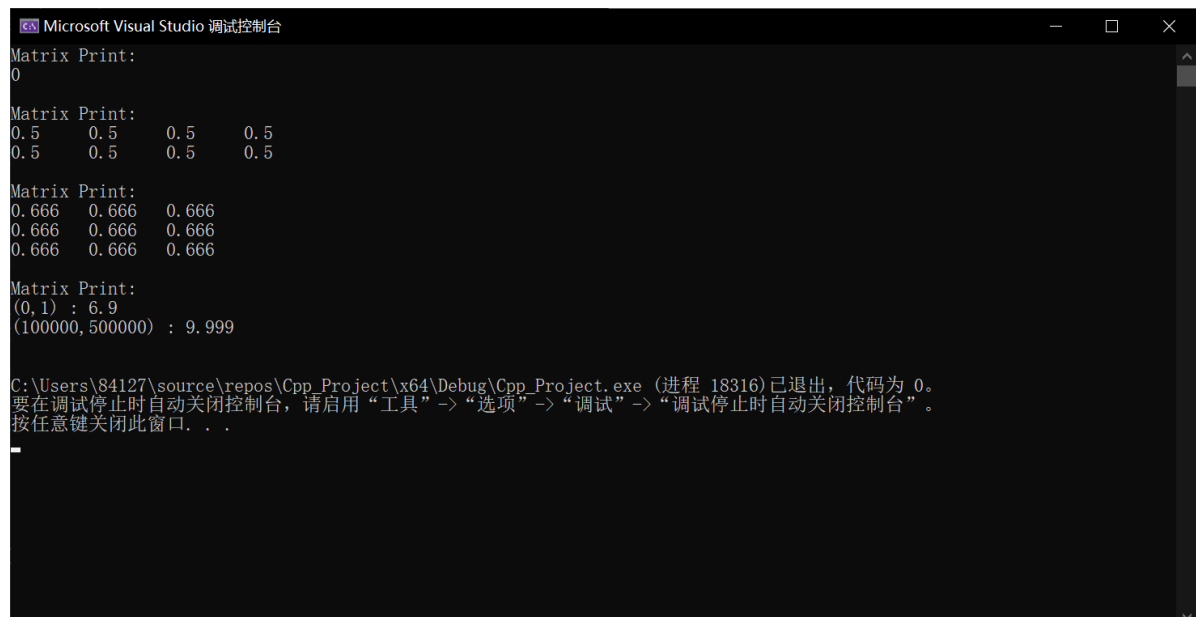
```
void test_1() {
    // create 1 * 1 matrix<int>
    Matrix<int> m_1 = Matrix<int>(1, 1);
    cout << m_1 << endl;

    // create 2*4 matrix<float> with initial value 0.5
    Matrix<float> m_2 = Matrix<float>(2, 4, 0.5);
    cout << m_2 << endl;

    // create 3 * 3 matrix<double> with initial vecator<vector<double>>
    vector<vector<double>> vec_1 = vector<vector<double>>(3, vector<double>(3,
0.666));
    Matrix<double> m_3 = Matrix<double>(vec_1);
    cout << m_3 << endl;

    // create sparse matrix<double> with 100000 * 500000
    map<pair<int, int>, double> m;
    m[{0, 1}] = 6.9;
    m[{100000, 500000}] = 9.999;
    Matrix<double> m_4 = Matrix<double>(m);
    cout << m_4 << endl;
}
```

Output:



```
Microsoft Visual Studio 调试控制台
Matrix Print:
1

Matrix Print:
0.5  0.5  0.5  0.5
0.5  0.5  0.5  0.5

Matrix Print:
0.666  0.666  0.666
0.666  0.666  0.666
0.666  0.666  0.666

Matrix Print:
(0, 1) : 6.9
(100000, 500000) : 9.999

C:\Users\84127\source\repos\Cpp_Project\x64\Debug\Cpp_Project.exe (进程 18316) 已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . .
```

Part 3.2

2) It supports all standard numeric types, including `std::complex`, integers, and is easily extensible to custom numeric types. (10 points)←

Test Code:

```
class user_datatype {
public:
    string value;
    user_datatype(string val) :value(val) {}
};

ostream& operator<<(ostream& os, const user_datatype& v) {
    cout << v.value;
    return os;
}

void test_2() {
    Matrix<char> m_1 = Matrix<char>(2, 2, 'x');
    cout << "Matrix<char>:" << endl;
    cout << m_1 << endl;

    Matrix<short> m_2 = Matrix<short>(2, 2, 2);
    cout << "Matrix<short>:" << endl;
    cout << m_2 << endl;

    Matrix<int> m_3 = Matrix<int>(2, 2, 3);
    cout << "Matrix<int>:" << endl;
    cout << m_3 << endl;

    Matrix<long> m_4 = Matrix<long>(2, 2, 4);
    cout << "Matrix<long>:" << endl;
    cout << m_4 << endl;

    Matrix<float> m_5 = Matrix<float>(2, 2, 5);
    cout << "Matrix<float>:" << endl;
    cout << m_5 << endl;

    Matrix<double> m_6 = Matrix<double>(2, 2, 6);
    cout << "Matrix<double>:" << endl;
    cout << m_6 << endl;

    Matrix<std::complex<double>> m_7 = Matrix<std::complex<double>>(2, 2, {1,
2});
    cout << "Matrix<std::complex<double>>:" << endl;
    cout << m_7 << endl;

    Matrix<user_datatype> m_8 = Matrix<user_datatype>(2, 2,
user_datatype("user_datatype"));
    cout << "Matrix<user_datatype>:" << endl;
    cout << m_8 << endl;
}
```

```
Microsoft Visual Studio 调试控制台
Matrix<char>:
Matrix Print:
x      x
x      x

Matrix<short>:
Matrix Print:
2      2
2      2

Matrix<int>:
Matrix Print:
3      3
3      3

Matrix<long>:
Matrix Print:
4      4
4      4

Matrix<float>:
Matrix Print:
5      5
5      5

Matrix<double>:
Matrix Print:
6      6
6      6

Matrix<std::complex<double>>:
Matrix Print:
(1,2)  (1,2)
(1,2)  (1,2)

Matrix<user_datatype>:
Matrix Print:
user_datatype  user_datatype
user_datatype  user_datatype

C:\Users\S4127\source\repos\Cpp_Project\x64\Debug\Cpp_Project.exe (进程 19864) 已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

Part 3.3

3) It supports matrix and vector arithmetic, including addition, subtraction, scalar multiplication, scalar division, transposition, conjugation, element-wise multiplication, matrix-matrix multiplication, matrix-vector multiplication, dot product and cross product. (20 points)↵

Test Code:

```
void test_3() {
    Matrix<double> m_1(3, 3, 1.2);
    Matrix<double> m_2(3, 3, 2.1);
    Matrix<double> m_4(3, 1, 0.5);
    Matrix<complex<double>> m_3(3, 3, { 1,3 });
    double number = 0.5;

    cout << "Matrix<double> m_1:" << endl;
    cout << m_1 << endl;
    cout << "Matrix<double> m_2:" << endl;
    cout << m_2 << endl;
    cout << "Matrix<complex<double>> m_3:" << endl;
    cout << m_3 << endl;
    cout << "Matrix<complex<double>> m_4:" << endl;
    cout << m_4 << endl;
    cout << "double number:" << endl;
    cout << number << endl;

    // test matrix addition
    cout << "(m_1 + m_2)" << endl;
    cout << (m_1 + m_2) << endl;
```

```

// test scalar addition
cout << "(m_1 + number)" << endl;
cout << (m_1 + number) << endl;

// test matrix subtraction
cout << "(m_1 - m_2)" << endl;
cout << (m_1 - m_2) << endl;

// test scalar subtraction
cout << "(m_1 - number)" << endl;
cout << (m_1 - number) << endl;

// test scalar multiplication
cout << "(m_1 * number)" << endl;
cout << (m_1 * number) << endl;

// test matrix multiplication
cout << "(m_1 * m_2)" << endl;
cout << (m_1 * m_2) << endl;

// test scalar division
cout << "(m_1 / number)" << endl;
cout << (m_1 / number) << endl;

// test matrix transposition
cout << "m_1.transposition()" << endl;
cout << m_1.transposition() << endl;

// test matrix conjugation
cout << "Before conjugation:" << endl;
cout << m_3 << endl;
cout << "After conjugation:" << endl;
cout << m_3.conjugation() << endl;

// test element-wise multiplication
cout << "m_1 .* m_2 (element-wise multiplication)" << endl;
cout << m_1.dot(m_2) << endl;

// test matrix-vector multiplication
cout << "m_4 vector-dot m_4" << endl;
cout << m_4.vector_dot(m_4) << endl;

// test cross-product
cout << "m_4 corss-product m_4" << endl;
cout << m_4.cross_product(m_4) << endl;

```

```

}

```

Matrix<double> m_1:

Matrix Print:

```
1.2    1.2    1.2
1.2    1.2    1.2
1.2    1.2    1.2
```

Matrix<double> m_2:

Matrix Print:

```
2.1    2.1    2.1
2.1    2.1    2.1
2.1    2.1    2.1
```

Matrix<complex<double>> m_3:

Matrix Print:

```
(1, 3)  (1, 3)  (1, 3)
(1, 3)  (1, 3)  (1, 3)
(1, 3)  (1, 3)  (1, 3)
```

Matrix<complex<double>> m_4:

Matrix Print:

```
0.5
0.5
0.5
```

double number:

```
0.5
```

(m_1 + m_2)

Matrix Print:

```
3.3    3.3    3.3
3.3    3.3    3.3
3.3    3.3    3.3
```

(m_1 + number)

Matrix Print:

```
1.7    1.7    1.7
1.7    1.7    1.7
1.7    1.7    1.7
```

(m_1 - m_2)

Matrix Print:

```
-0.9   -0.9   -0.9
-0.9   -0.9   -0.9
-0.9   -0.9   -0.9
```

(m_1 - number)

Matrix Print:

```
0.7    0.7    0.7
0.7    0.7    0.7
0.7    0.7    0.7
```

(m_1 * number)

```
Matrix Print:
0.6      0.6      0.6
0.6      0.6      0.6
0.6      0.6      0.6
```

```
(m_1 * m_2)
Matrix Print:
7.56     7.56     7.56
7.56     7.56     7.56
7.56     7.56     7.56
```

```
(m_1 / number)
Matrix Print:
2.4      2.4      2.4
2.4      2.4      2.4
2.4      2.4      2.4
```

```
m_1.transposition()
Matrix Print:
1.2      1.2      1.2
1.2      1.2      1.2
1.2      1.2      1.2
```

```
Before conjugation:
Matrix Print:
(1, 3)   (1, 3)   (1, 3)
(1, 3)   (1, 3)   (1, 3)
```

```
m_1 .* m_2 (element-wise multiplication)
Matrix Print:
2.52     2.52     2.52
2.52     2.52     2.52
2.52     2.52     2.52
```

```
m_4 vector-dot m_4
0.75
m_4 corss-product m_4
Matrix Print:
0        0        0
```

Part 3.4

4) It supports basic arithmetic reduction operations, including finding the maximum value, finding the minimum value, summing all items, calculating the average value (all supporting axis-specific and all items). (10 points)↵

Test Code:

```

void test_4() {
    vector<vector<double>> v(3, vector<double>(3));
    v[0][0] = 2;
    v[0][1] = 3;
    v[0][2] = 1;
    v[1][0] = v[1][1] = v[1][2] = 1;
    v[2][0] = 3;
    v[2][1] = 5;
    v[2][2] = 2;
    Matrix<double> m1(v);

    cout << "m1.row_max(1)" << endl;
    cout << m1.row_max(1) << endl;
    cout << "m1.col_min(2)" << endl;
    cout << m1.col_min(2) << endl;
    cout << "m1.cols_sum()" << endl;
    cout << m1.cols_sum() << endl;
    cout << "m1.matrix_avg()" << endl;
    cout << m1.matrix_avg() << endl;
}

```

```

m1.row_max(1)
3
m1.col_min(2)
1
m1.cols_sum()
Matrix Print:
6      9      4

m1.matrix_avg()
2.11111

```

Part 3.5

5) It supports computing eigenvalues and eigenvectors, calculating traces, computing inverse and computing determinant. (10 points)←

Test Code:

```

void test_5() {
    vector<vector<double>> v(3, vector<double>(3));
    v[0][0] = 2;
    v[0][1] = 3;
    v[0][2] = 1;
    v[1][0] = v[1][1] = v[1][2] = 1;
    v[2][0] = 3;
    v[2][1] = 5;
    v[2][2] = 2;
    Matrix<double> m1(v);
    cout << "m1.getEigenValues()" << endl;
    auto ev = m1.getEigenValues();
    for (size_t i = 0; i < ev.size(); i++)

```

```

{
    cout << ev[i] << " ";
}
cout << endl;
cout << "m1.getEigenVectors()" << endl;
cout << m1.getEigenVectors() << endl;
cout << "m1.trace()" << endl;
cout << m1.trace() << endl;
cout << "m1.inverse()" << endl;
cout << m1.inverse() << endl;
cout << "m1.determinant()" << endl;
cout << m1.determinant() << endl;
}

```

```

1 m1.getEigenValues()
2 5.5114 -0.752503 0.241098
3 m1.getEigenVectors()
4 Matrix Print:
5 0.483195      0.767653      0.767653
6 0.290212      -0.250497     -0.250497
7 0.826014      -0.589881     -0.589881
8
9 m1.trace()
10 5
11 m1.inverse()
12 Matrix Print:
13 3      1      -2
14 -1     -1      1
15 -2      1      1
16
17 m1.determinant()
18 -1

```

Part 3.6

6) It supports the operations of reshape and slicing. (10 points) ←

Test Code:

```

void test_6() {
    vector<vector<double>> v(3, vector<double>(3));
    v[0][0] = 2;
    v[0][1] = 3;
    v[0][2] = 1;
    v[1][0] = v[1][1] = v[1][2] = 1;
    v[2][0] = 3;
    v[2][1] = 5;
    v[2][2] = 2;
}

```



```

Matrix<double> m1(v);

cout << "m1.reshape(1, 9)" << endl;
cout << m1.reshape(1, 9) << endl;

cout << "m1.slice({ 1,2 }, { 1,2 })" << endl;
cout << m1.slice({ 1,2 }, { 1,2 }) << endl;
}

```

```

Microsoft Visual Studio 调试控制台
m1.reshape(1, 9)
Matrix Print:
2 3 1 1 1 1 3 5 2

m1.slice({ 1,2 }, { 1,2 })
Matrix Print:
2 3
1
C:\Users\84127\source\repos\Cpp_Project\Debug\Cpp_Project.exe (进程 9480) 已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。

```

Part 3.7

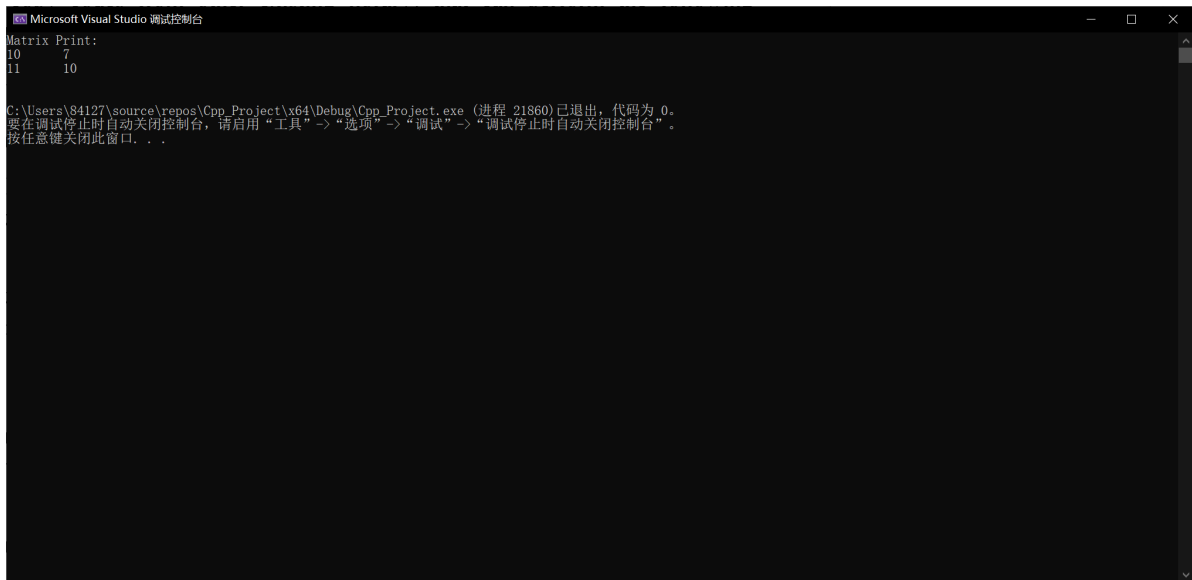
7) It supports convolutional operations of two matrices. (10 points)←

Test Code:

```

void test_7() {
    vector<vector<double>> v(3, vector<double>(3));
    v[0][0] = 2;
    v[0][1] = 3;
    v[0][2] = 1;
    v[1][0] = v[1][1] = v[1][2] = 1;
    v[2][0] = 3;
    v[2][1] = 5;
    v[2][2] = 2;
    Matrix<double> m1(v);
    vector<vector<double>> v2(2, vector<double>(2));
    v2[0][0] = 1;
    v2[0][1] = 2;
    v2[1][0] = 1;
    v2[1][1] = 1;
    Matrix<double> m2(v2);
    cout << m1.convolution(m2, 1, 0) << endl;
}

```



Part 3.8

8) It supports to transfer the matrix from OpenCV to the matrix of this library and vice versa. (10 points)←

Test Code:

```
void test_8() {
    vector<vector<double>> v(3, vector<double>(3));
    v[0][0] = 2;
    v[0][1] = 3;
    v[0][2] = 1;
    v[1][0] = v[1][1] = v[1][2] = 1;
    v[2][0] = 3;
    v[2][1] = 5;
    v[2][2] = 2;
    Matrix<double> m1(v);
    cout << "Matrix<double>:" << endl;
    cout << m1 << endl;
    Mat m = Matrix<double>::MatrixToMat_Double(m1);
    cout << "Mat:" << endl;
    cout << m << endl;

    Matrix<double> m2 = Matrix<double>::MatToMatrix_Double(m);
    cout << m2 << endl;
}
```

```
Microsoft Visual Studio 调试控制台
Matrix<double>:
Matrix Print:
2      3      1
1      1      1
3      5      2

Mat:
[2, 3, 1;
 1, 1, 1;
 3, 5, 2]
Matrix Print:
2      3      1
1      1      1
3      5      2

C:\Users\84127\source\repos\Cpp_Project\x64\Debug\Cpp_Project.exe (进程 4176) 已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . .
```

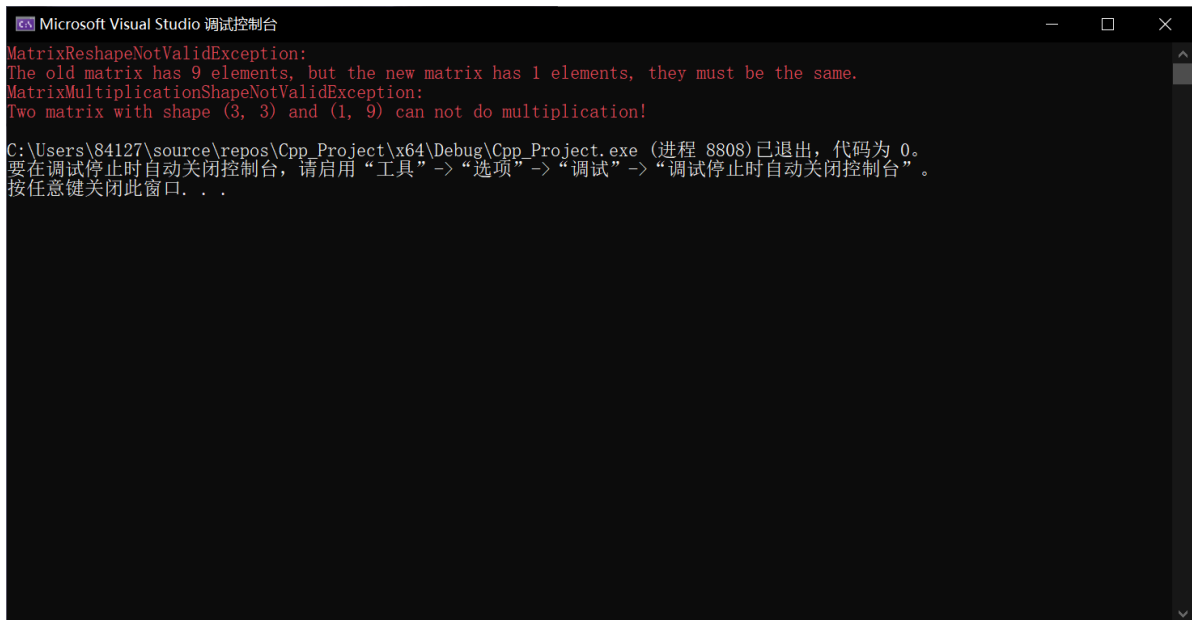
Part 3.9

9) It should process likely exceptions as much as possible. (10 points)←

Test Code:

```
void test_9() {
    vector<vector<double>> v(3, vector<double>(3));
    v[0][0] = 2;
    v[0][1] = 3;
    v[0][2] = 1;
    v[1][0] = v[1][1] = v[1][2] = 1;
    v[2][0] = 3;
    v[2][1] = 5;
    v[2][2] = 2;
    Matrix<double> m1(v);
    try {
        m1.reshape(1, 1);
    }
    catch (BaseException& e) {
        e.Show();
    }

    Matrix<double> m2(v);
    m2.reshape(1, 9, true);
    try {
        m1 * m2;
    } catch (BaseException& e) {
        e.Show();
    }
}
```

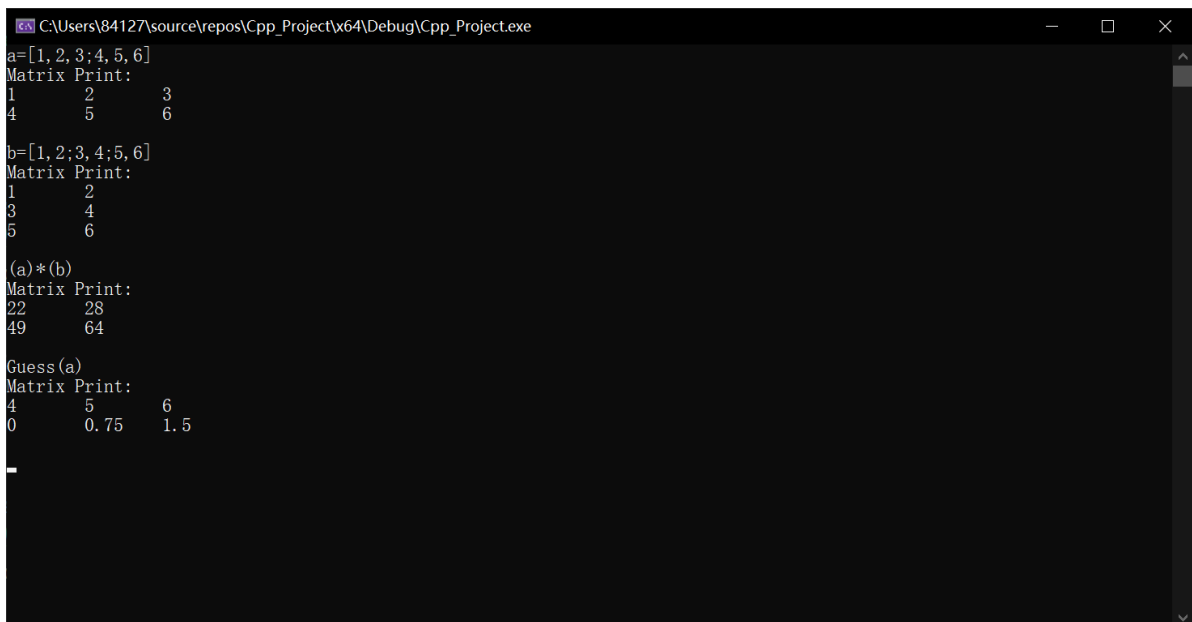


Part 3.10 BONUS

User Interaction

Test Code:

```
void test_10() {  
    SolveExpressions solver;  
    string str;  
    while (cin >> str) {  
        solver.init(str);  
        cout << solver.solve(0, str.size() - 1) << endl;  
    }  
}
```



Solve Linear Equations

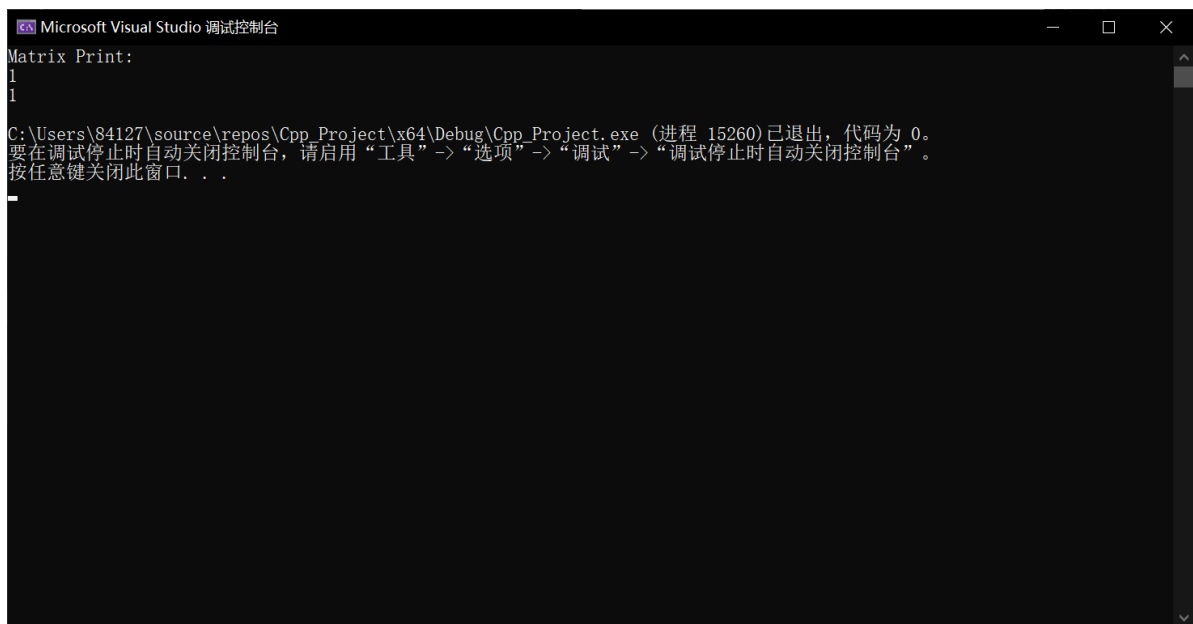
Test Code:

```
void test_11() {
```

```

vector<vector<double>> v(2, vector<double>(2));
v[0][0] = 1;
v[0][1] = 0;
v[1][0] = 1;
v[1][1] = 1;
vector<vector<double>> v1(2, vector<double>(1));
v1[0][0] = 1;
v1[1][0] = 2;
Matrix<double> n1(v), n2(v1);
try {
    cout << Matrix<double>::solveLinearEquations(n1, n2);
}
catch (BaseException& e) {
    e.Show();
}
}

```



Part 4 - Difficulties & Solutions

1.It's hard to make a matrix calculator

solution:match the brackets and recursion to solve

2.It's hard to find the inverse of matrix.

solution:use determinant and circulation

3.Hard to store so many types

solution:using template so that the class can has good compatibility

4.When store high dimensional tensor, we can not know how many dimensions it has when we define the class

solution:using a Type*, only one dimension pointer, and calculate the element's location by the parameters each time.

5.different size of matrix cannot do multiply add or many other operations.

solution:using exceptions when making these kind of wrong operators.

6.To protect the data, we make it private, but in some functions we must use it.

solution:use member functions or friend functions.

7.it hard to store different size of matrix

solution:using vector, and we can get the size at the same time

