code1 实验报告

陈泽高 PB20000302

2023 年 3 月 12 日

1 实验介绍

本次实验要求用 C++ 实现矩阵类 **matrix**：该类用一维数组和两个整数分别存储矩阵的元素、矩阵的行数及列数，能够调用赋值、加减乘（除）法运算、切片操作、输出矩阵元素等方法。

2 算法与实现

要求实现的功能并不复杂，主要有三大部分：

```
基础的赋值功能、重载等号运算符
'+','-','*','/'等运算
输出结果，如打印矩阵值，获取矩阵元素、子阵，输出错误信息等
```

其中矩阵的基础运算直接套用公式即可。需要注意几点：

1. 由于类中包含指针成员，需要对指针指向的内容进行拷贝，称为“深拷贝”，默认的等号运算符是“浅拷贝”。

2. 有时我们会在方法 (操作符) 中定义变量或者分配内存，然后需要返回变量值 (类的实例) 或者保留分配的内存。该情况下不能直接返回它们的引用，因为在跳出该方法时，在其中定义的变量 (类的实例) 生命周期结束，会被自动清理 (析构)，即无法返回有效内容。对于本次实验，应该返回'Matrix<Tem>' 类的实例，这样实例就能被拷贝并带出 (定义该实例的) 方法。分配内存的指针同理，在跳出方法前，在方法中定义的指针不能指向要保留的内存空间。

3 测试结果

主要进行了正确性测试1和效率测试2，前者对编写的各方法以及不同模板进行了测试，后者对比了 Matrix 类与 Eigen 库的 MatrixXd 类在进行较大规模矩阵乘法的运算速度 (矩阵乘法的时间复杂度最高，理论上应该用它来测试效率)，实验结果如图所示，源代码详见附件或本文附录。

4 实验分析

本次实验完成了 Matrix 类的实现与测试。目标类在测试函数中正常运行。不难看出，在没有进行优化 (debug) 的情况下，Matrix 类明显快于 Eigen::MatrixXd 类，这是因为我们自己实现的

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
整型数据 准确性测试:
this matrix has size (4 x 3)
the entries are:
4       11      4
11      2       16
5       18      13
21      10      4
A: this matrix has size (3 x 4)
the entries are:
0       1       2       3
1       2       3       4
2       3       4       5
B: this matrix has size (4 x 3)
the entries are:
0       -1      -2
1       0       -1
2       1       0
3       2       1
R: this matrix has size (4 x 3)
the entries are:
4       12      6
10      2       17
3       17      13
18      8       3
C: this matrix has size (3 x 3)
the entries are:
14      8       2
20      10      0
26      12      -2
D: this matrix has size (4 x 4)
the entries are:
-5      -8      -11     -14
-2      -2      -2      -2
1       4       7       10
4       10      16      22
E: this matrix has size (1 x 3)
the entries are:
85      68      53
F: this matrix has size (3 x 2)
the entries are:
1       2
2       3
3       4
```

(a) 整型

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
双精度数据 准确性测试:
this matrix has size (4 x 3)
the entries are:
3.4     10.6    17.4
13.8    19.8    3.6
2       14.2    14.4
22.4    20      8.2
A: this matrix has size (3 x 4)
the entries are:
0       1       2       3
1       2       3       4
2       3       4       5
B: this matrix has size (4 x 3)
the entries are:
0       -1      -2
1       0       -1
2       1       0
3       2       1
R: this matrix has size (4 x 3)
the entries are:
3.4     11.6    19.4
12.8    19.8    4.6
0       13.2    14.4
19.4    18      7.2
C: this matrix has size (3 x 3)
the entries are:
14      8       2
20      10      0
26      12      -2
D: this matrix has size (4 x 4)
the entries are:
-5      -8      -11     -14
-2      -2      -2      -2
1       4       7       10
4       10      16      22
E: this matrix has size (1 x 3)
the entries are:
86      108.2   56
F: this matrix has size (3 x 2)
the entries are:
1       2
2       3
3       4
```

(b) 双精度型

图 1: 正确性测试

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Matrix 效率测试:
My Matrix, runtime: 0.0894401
Eigen's Matrix, runtime: 0.0120396
```

(a) 优化 (Release)

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Matrix 效率测试:
My Matrix, runtime: 2.03855
Eigen's Matrix, runtime: 3.73207
```

(b) 无优化 (Debug)

图 2: 效率测试

类会对数据直接操作；而在优化 (Release) 编译后，同一程序的运算速度都大幅提升，而且 Eigen 库的实现大幅反超了笔者自己的实现（情理之中）。

5 附件内容

本次作业附件为：

```
matrix_main.cpp  :源代码
homework1/  :用VS构建的项目， 使用前需调整属性
```

A 代码

包含 Matrix 类、Assert 函数 (检查与报错)、两个 test 函数、主函数 (测试用)、系数矩阵类 (To Be Done), 其中部分乱码应该是中文编码的问题，暂未找到解决方法，不过不影响查看（详见附件）

```
//#include <cstdio>
#include <chrono>
#include <iostream>
#include <vector>

#include "Eigen/Dense"
//#include <Eigen/QR>

bool Assert(bool state);

// todo 5: change the class in to a template class
//#define Tem double
template<typename Tem>
class Matrix {
private:
int rows, cols;
Tem *data;

public:
    // default constructor
    // https://en.cppreference.com/w/cpp/language/constructor
Matrix() :rows(1), cols(1), data(new Tem[1]) { data[0] = 0; } // Matrix() = delete;

    // constructor with initilizer list
Matrix(int r, int c)
: rows(r), cols(c), data(new Tem[r*c]) {}
Matrix(const Matrix& Mat) //   ú £Mat¶¼¿   ø
: rows(Mat.rows), cols(Mat.cols), data(new Tem[Mat.rows * Mat.cols])
{
// std::cout << "Created!" << std::endl;
for (int n = 0; n < rows*cols; n++)
data[n] = Mat[n];
}

    // desctructor
    // https://en.cppreference.com/w/cpp/language/destructor
~Matrix() { if(data!=nullptr)delete data; }

    int nrow() const {return rows;}
    int ncol() const {return cols;}

    // operator overloding
```

```
    Tem& operator()(int r, int c) {return data[r*cols + c]; } // ¿ ı  ġ°  á±
Tem& operator()(int r, int c) const { return data[r*cols + c]; }// ²»¸ı  µġ°  á±
Tem& operator[](int n) { return data[n]; /* todo 3: particular entry of the matrix*/ }
    Tem& operator[](int n) const {return data[n]; /* todo 3: particular entry of the matrix*/}

    Matrix col(int col)
{/* todo 4: particular column of the matrix*/
Matrix Mat_Col(rows, 1); // » »  ú¯ ½  ±» ¿²»»  I · µ»   ¸£¬²»
for (int i = 0; i < rows; i++)
Mat_Col[i] = data[i*cols+col];
return Mat_Col;
}
    Matrix row(int row)
{/* todo 4: particular row of the matrix*/
Matrix Mat_Row = Matrix(1, cols);
for (int i = 0; i < cols; i++)
Mat_Row(0, i) = (*this)(row,i);// data[row*cols + i];
return Mat_Row;
}
    Matrix submat(int rowL, int rowU, int colL, int colU) const
{/* todo 4: return a sub-matrix specified by the input parameters*/
Assert((rowL<=rowU) && (colL<=colU));
Matrix Mat_Sub(rowU-rowL+1, colU-colL+1);
for (int i = 0; i < rowU - rowL + 1; i++)
{
for (int j = 0; j < colU - colL + 1; j++)
Mat_Sub(i, j) = (*this)(i + rowL, j + colL);//data[(i + rowL)*cols + (j + colL)];
}
return Mat_Sub;
}

    // constant alias
Matrix& operator= (const Matrix& rhs)
{
// std::cout << "'=' called\n ~~~~~~~~~~~" << std::endl;
if(data!=nullptr) delete[] data;
Tem* newData = new Tem[rhs.cols*rhs.rows];
rows = rhs.rows; cols = rhs.cols;
for (int i = 0; i < cols*rows; i++)
newData[i] = rhs.data[i];

data = newData;
return *this;
}

/*      I · µ» µÍ́ô¦  ½±    +=¡¢−= µ    µ
     ± (A*B).print()       Ï · µ»  Ǿ¿± µ    */
```

```
    Matrix operator+ (const Matrix& rhs)
{
Assert(cols == rhs.ncol() && rows == rhs.nrow());
Matrix res(rows, cols);
for (int n = 0; n < cols*rows; n++)
res[n] = (*this)[n] + rhs[n];
return res;
}
    Matrix operator- (const Matrix& rhs)
{
Assert(cols == rhs.ncol() && rows == rhs.nrow());
Matrix res(rows, cols);
for (int n = 0; n < cols*rows; n++)
res[n] = (*this)[n] - rhs[n];
return res;
}

// ¾  ¨
    Matrix operator* (const Matrix& rhs)
{
Assert(cols == rhs.nrow());
Matrix res(rows, rhs.ncol());
for(int i=0; i<rows; i++)
for (int j = 0; j < rhs.ncol(); j++)
{
res(i, j) = 0;
for (int k = 0; k < cols; k++)
res(i, j) += (*this)(i, k) * rhs(k, j);
}
// res.print(); //test
return res;
}

  // ¶      £¬ F °  0  Ĩ    º
Matrix operator/ (const Matrix& rhs)
{
Assert(rows == rhs.nrow() && cols == rhs.ncol());
Matrix res(rows, cols);
for (int n = 0; n < cols*rows; n++)
res[n] = (*this)[n] + rhs[n];
return res;
}

    Matrix operator+= (const Matrix& rhs)
{
Matrix res(*this + rhs);
*this = res;
```

```
return *this;
}

    Matrix operator-= (const Matrix& rhs)
{
Matrix res(*this - rhs);
*this = res;
return *this;
}

    Matrix operator*= (const Matrix& rhs)
{
Matrix res(*this * rhs);
*this = res;
return *this;
}
Matrix operator/= (const Matrix& rhs)
{
Matrix res(*this / rhs);
*this = res;
return *this;
}

Matrix operator+ (Tem v)
{
Matrix res(rows, cols);
for (int n = 0; n < cols*rows; n++)
res[n] = (*this)[n] + v;
return res;
}
    Matrix operator- (Tem v)
{
Matrix res(rows, cols);
for (int n = 0; n < cols*rows; n++)
res[n] = (*this)[n] - v;
return res;
}
    Matrix operator* (Tem v)
{
Matrix res(rows, cols);
for (int n = 0; n < cols*rows; n++)
res[n] = (*this)[n] * v;
return res;
}
    Matrix operator/ (Tem v)
{
Assert(fabs(v) < 1e-10); //   v²»½ 0
```

```
Matrix res(rows, cols);
for (int n = 0; n < cols*rows; n++)
res[n] = (*this)[n] / v;
return res;
}

Matrix operator+= (Tem v)
{
Matrix res(*this + v);
*this = res;
return *this;
}
Matrix operator-= (Tem v)
{
Matrix res(*this - v);
*this = res;
return *this;
}
    Matrix operator*= (Tem v)
{
Matrix res(*this * v);
*this = res;
return *this;
}
    Matrix operator/= (Tem v)
{
Matrix res(*this / v);
*this = res;
return *this;
}

    void print () const {
        printf("this matrix has size (%d x %d)\n", rows, cols);
        printf("the entries are:\n");
for (int i = 0; i < rows; i++)
{
for (int j = 0; j < cols; j++)
std::cout << (*this)(i, j) << '\t';
std::cout << std::endl;
}
    }

};

bool Assert(bool state)
{
if (state)
```

```
return true;
else
std::cout << "Err!" << std::endl;
exit(0);
}


// BONUS: write a sparse matrix class
template<typename R>
class SparseMatrix {

};

template<typename type>
void test_1()
{
std::cout << ":  ȷ  " << std::endl; // ?<< static_cast<char*>(type)

Matrix<type> A(3, 4), B(4, 3), R(4, 3); // ¶¨  Ľ¨¹¹º¯

for (int i = 0; i < A.nrow(); i++) //  ·½·¨ nrow(), ncol()
for (int j = 0; j < A.ncol(); j++)
A(i, j) = i + j; //      ()''
for (int i = 0; i < B.nrow(); i++)
for (int j = 0; j < B.ncol(); j++)
{
B(i, j) = i - j;
R(i, j) = (rand() % 100) / static_cast<type>(5);
}

Matrix<type> C = A * B; // ¹¹ 🄵  Matrix(const Matrix& Mat)
Matrix<type> D; D = B * A; // Ĭ ¨¹¹º¯ £¬      '='
Matrix<type> E(B.row(1)); // º¯  row(), col() ,¶
E += A.row(0)*(B + R); //       '+=' '-='µ   ¸´
Matrix<type> F(A.submat(0, 2, 1, 2)); // º¯  submat()     F=A[1:2, 0:3]

(B + R).print(); //       '*'   Matrix

std::cout << "A: "; A.print(); // º¯  print() £¬
std::cout << "B: "; B.print();
std::cout << "R: "; R.print();
std::cout << "C: "; C.print();
std::cout << "D: "; D.print();
std::cout << "E: "; E.print();
std::cout << "F: "; F.print();
}
```

```
void test_2()
{
std::cout << "   :" << std::endl;
int n_rows = 300, n_cols = 200;
auto start = std::chrono::steady_clock::now();
{
Matrix<double> A(n_rows, n_cols), B(n_cols, n_rows), R(n_cols, n_rows);

for (int i = 0; i < A.nrow(); i++)
for (int j = 0; j < A.ncol(); j++)
A(i, j) = i + j;
for (int i = 0; i < B.nrow(); i++)
for (int j = 0; j < B.ncol(); j++)
{
B(i, j) = i - j;
R(i, j) = (rand() % 100) / 9.7;
}
Matrix<double> C = A * B;
Matrix<double> D; D = R * A;
}
auto end = std::chrono::steady_clock::now();
std::chrono::duration<double> runTime = end - start;
std::cout << "My Matrix, runtime: " << runTime.count() << std::endl;

// todo 8: use Eigen and compare
start = std::chrono::steady_clock::now();
{
Eigen::MatrixXd A(n_rows, n_cols), B(n_cols, n_rows), R(n_cols, n_rows);
for (int i = 0; i < A.rows(); i++)
for (int j = 0; j < A.cols(); j++)
A(i, j) = i + j;
for (int i = 0; i < B.rows(); i++)
for (int j = 0; j < B.cols(); j++)
{
B(i, j) = i - j;
R(i, j) = (rand() % 100) / 9.7;
}

Eigen::MatrixXd C = A * B;
Eigen::MatrixXd D(n_cols, n_cols); D = R * A;
}
end = std::chrono::steady_clock::now();
runTime = end - start;
std::cout << "Eigen's Matrix, runtime: " << runTime.count() << std::endl;
}

int main()
```

```
{
srand(time(0));
/* test 1:  ȷ    */
std::cout << "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
std::cout << "    ¾  "; test_1<int>();

std::cout << "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
std::cout << "🄵¾«¶  ¾  "; test_1<double>();

/* test 2      */
std::cout << "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
std::cout << "Matrix "; test_2();

    return 0;
}
```