Report

PB20020480 王润泽

1 问题描述

C++编程环境下，自定义矩阵类，稀疏矩阵类，并与Eigen库中的矩阵实现进行对比

2 实验内容

2.1 简单稠密矩阵

2.1.1成员变量

该矩阵类模板包含以下成员变量：

- `rows`：表示矩阵的行数
- `cols`：表示矩阵的列数
- `size`：表示矩阵的大小（即行数乘以列数）
- `data`：表示存储矩阵数据的数组指针
- `data_transpose`：表示存储矩阵转置数据的数组指针

2.1.2成员函数

该矩阵类模板包含以下成员函数：

构造函数

- `Matrix()`：默认构造函数，将矩阵的行数、列数、大小均初始化为0，`data` 和 `data_transpose` 初始化为 `nullptr`。
- `Matrix(int r, int c)`：构造函数，用于创建一个 r 行 c 列的矩阵，并将所有元素初始化为0。
- `Matrix(int r, int c, T* d)`：构造函数，用于从指针 `d` 初始化一个 r 行 c 列的矩阵。
- `Matrix(const Matrix& rhs)`：拷贝构造函数，用于从一个已有的矩阵 `rhs` 中创建一个新的矩阵。

析构函数

- `~Matrix()`：析构函数，用于释放矩阵所占用的内存。

成员函数

- `int nrow() const`：返回矩阵的行数。
- `int ncol() const`：返回矩阵的列数。
- `void setZeros(int r, int c)`：重新设置矩阵的行数和列数，并将所有元素初始化为0。
- `T& operator()(int r, int c) const`：用于访问矩阵中第 r 行第 c 列的元素。
- `T& operator[](int n)`：用于访问矩阵中第 n 个元素。
- `Matrix col(int c)`：用于返回矩阵中第 c 列。

- `Matrix row(int r)`：用于返回矩阵中第 r 行。
- `Matrix submat(int startRow, int startCol, int numRows, int numCols) const`：用于返回由输入参数指定的子矩阵。

操作符重载

该矩阵类模板还包含以下操作符重载：

- `Matrix& operator=(const Matrix& rhs)`：用于将一个矩阵 `rhs` 赋值给另一个矩阵。
- `Matrix operator+(const Matrix& rhs) const`： 用于实现矩阵加法

  `Matrix operator+(T v)`：用于实现矩阵与常量加法
- `Matrix operator*(const Matrix& rhs) const`：用于实现矩阵乘法，返回一个新的矩阵作为结果。
- `Matrix operator/(const Matrix& rhs) const`：用于实现矩阵按位除法运算，返回一个新的矩阵作为结果。

剩下的还包括 `-，-=；+=，*=，/=` 等常见运算

输出

- `print` 对外打印矩阵内容

2.1.3 程序输出

```
A Matrix:
this matrix has size (3 x 3)
the entries are:
0 1 2
3 4 5
6 7 8

B Matrix:
this matrix has size (3 x 3)
the entries are:
1 2 3
4 5 6
7 8 9

A sub Matrix:
this matrix has size (2 x 2)
the entries are:
0 1
3 4
```

```
A+B =
this matrix has size (3 x 3)
the entries are:
1 3 5
7 9 11
13 15 17

A-B =
this matrix has size (3 x 3)
the entries are:
-1 -1 -1
-1 -1 -1
-1 -1 -1

AxB =
this matrix has size (3 x 3)
the entries are:
18 21 24
54 66 78
90 111 132

A/B =
this matrix has size (3 x 3)
the entries are:
0 0.5 0.666667
0.75 0.8 0.833333
0.857143 0.875 0.888889
```

图1：稠密矩阵基本输出

2.1.4 程序健壮性

由于在进行矩阵运算索引时，难免会出现数组越界、除以零等错误操作，故在代码中增加了异常处理的提醒

```
try{cout << A(3, 0) << std::endl;
} catch (std::out_of_range& ex) {  std::cerr << "Error: " << ex.what() << std::endl;
} catch (std::exception& ex) { std::cerr << "Error: " << ex.what() << std::endl;}
```

```
Error: out of range
```

2.2 稀疏矩阵

SparseMatrix类是一个稀疏矩阵类，其中元素的大多数值为0，只有很少的非零元素。与常规矩阵不同，稀疏矩阵中非零元素的数量较少，因此存储和处理这种矩阵需要特殊的数据结构和算法。SparseMatrix类提供了一种简单而高效的方法来存储和处理稀疏矩阵。

2.2.0 三元组

稀疏矩阵主要采用Triplet的三元组结构，用来描述矩阵的信息

```
template <typename T>
struct Triplet {
    int row, col;
    T value;
    Triplet(int row, int col, T value)
        : row(row), col(col), value(value) {
    }
    Triplet() = default;  // 提供默认构造函数
};
```

`SparseMatrix`是一个稀疏矩阵类，它是一个用三元组表示的矩阵，能够进行转置、压缩等操作，并且支持矩阵乘法。

2.2.1 成员变量

该矩阵类模板包含以下成员变量：

- `int rows ,cols` 矩阵行列
- `std::vector<Triplet<T>> triplets`； 三元组内容表示
- `std::vector<int> outer_starts` 每行元素的起始位置
- `std::vector<int> inner_indices` 非零元素在列中的位置
- `std::vector<T> values`： 非零元素值
- `std::vector<int> row_nnz`：每行非零元素的个数
- `std::vector<int> col_nnz`：每列非零元素的个数

2.2.2 成员函数

基本函数

- `void insert(int row, int col, T value);`添加一个元素
- `SparseMatrix<T> transpose() const;`转置矩阵
- `void transposeInplace() const;`就地转置
- `std::vector<Triplet<T>> row_triple(int r) const;`返回第r行的三元组
- `void makeCompressed();`压缩矩阵

- `void setFromTriplets(const std::vector<Triplet<T>>& t);` 从三元组构造稀疏矩阵

运算符重载

- `SparseMatrix<T>& operator=(const SparseMatrix<T>& rhs)` 赋值
- `SparseMatrix<T> operator*(const SparseMatrix<T>& rhs) const` 矩阵乘法
- `T operator()(int row, int col)` 矩阵索引

输出

- `void printSparse() const` 输出稀疏矩阵格式
- `void print() const` 输出完整矩阵格式

2.2.3 程序输出

```
A Sparse Mat:
this matrix has size (8 x 8)
the entries are:
(0, 0) 1
(0, 2) 4
(1, 1) 2
(2, 2) 3
(3, 3) 5
(4, 4) 6
(5, 5) 7
(6, 6) 8
(7, 7) 9
```

```
B Sparse Mat:
this matrix has size (8 x 8)
the entries are:
1 0 4 0 0 0 0 0
0 2 0 0 0 0 0 0
0 0 3 0 0 0 0 0
0 0 0 5 0 0 0 0
0 0 0 0 6 0 0 0
0 0 0 0 0 7 0 0
0 0 0 0 0 0 8 0
0 0 0 0 0 0 0 9
```

```
AxB=
this matrix has size (8 x 8)
the entries are:
1 0 16 0 0 0 0 0
0 4 0 0 0 0 0 0
0 0 9 0 0 0 0 0
0 0 0 25 0 0 0 0
0 0 0 0 36 0 0 0
0 0 0 0 0 49 0 0
0 0 0 0 0 0 64 0
0 0 0 0 0 0 0 81
```

图2：稀疏矩阵输出

2.3 与Eigen 库比较

简单使用Eigen库的稠密矩阵进行1000x1000乘法比较如下

```
My Matrix Multi time(1000x1000):2714 ms
Eigen Matrix Multi time(1000x1000):132 ms
```

由于Eigen库由编译优化操作，同时Eigen库使用了一些高效的矩阵乘法算法，例如基于BLAS和LAPACK的算法，可以充分利用现代CPU的特殊指令集，如SSE和AVX；并且Eigen库的矩阵和向量数据结构采用了特殊的优化，可以使得数据在内存中的存储方式更加紧凑和连续，从而可以更好地利用CPU的缓存。

综合以上原因，所以自己的定义的矩阵乘法运算较为缓慢。

3 总结

本次实验有以下收获总结：

- 熟悉了关于C++模板类的定义，自我定义了一个矩阵类，并进行简单运算和异常处理；
- 仿照Eigen库的形式，实现了稀疏矩阵类，对大规模稀疏矩阵创建与运算操作的有了更多的理解
- 用自己定义的简单矩阵乘法与Eigen库所提供的矩阵乘法比较，发现具有明显差异，对Eigen库中运算的优化有了更加深刻的了解