1.3数组类和系数的操作

与矩阵类只适用与线性代数运算相反，数组类提供通用的数组类，能不利用线性代数的知识来对系数进行操作，比如对每个系数加上一个常数，或者乘上两个数组的系数。

1.数组类型

跟矩阵类一样，数组也是一个具有相同模板参数的类模板。和矩阵类相同，前三个模板参数是必须提供实参的：

Array<typename Scalar, int RowsAtCompileTime, int ColsAtCompileTime>

后三个模板参数是选择性的。

对某些相同的情况，eigen同样也提供类型别名，但是与矩阵类有点不同的是，数组类用于一维和二维数组。依据惯例，ArrayNt表示一维数组，N表示数组大小，t表示元素类型。对二维数组来讲，使用ArrayNNt来表示，如下所示：

Array<float,Dynamic,1> ArrayXf //动态绑定的一维数组，元素类型为单精度浮点型

Array<float,3,1> Array3f //数组大小为3的一维数组，元素类型为单精度浮点型

Array<double,Dynamic,Dynamic> ArrayXXd//动态绑定的二维数组 ，元素类型为双精度浮点型

Array<double,3,3> Array33d //数组大小分别为3的二维数组，元素类型为双精度浮点型

2.访问数组元素

与矩阵一样，重载的括号运算符提供数组元素读写操作。同时<<输出运算符也能初始化数组或者打印它们。

#include <Eigen/Dense>

#include <iostream>

using namespace Eigen;

using namespace std;

int main()

{

ArrayXXf m(2,2);

// assign some values coefficient by coefficient

m(0,0) = 1.0; m(0,1) = 2.0;

m(1,0) = 3.0; m(1,1) = m(0,1) + m(1,0);

// print values to standard output

cout << m << endl << endl;

// using the comma-initializer is also allowed

m << 1.0,2.0,

3.0,4.0;

// print values to standard output

cout << m << endl;

}

//output

1 2

3 5

1 2

3 4

3.加减法

跟矩阵加减法一样，数组加减法只对两个数组有相同的大小有效。但是，数组能完成数组与标量的加法，这相当于数组内的所有元素都加上这个数值，这个提供了矩阵对象不能直接使用的功能。

#include <Eigen/Dense>

#include <iostream>

using namespace Eigen;

using namespace std;

int main()

{

ArrayXXf a(3,3);

ArrayXXf b(3,3);

a << 1,2,3,

4,5,6,

7,8,9;

b << 1,2,3,

1,2,3,

1,2,3;

// Adding two arrays两个数组相加

cout << "a + b = " << endl << a + b << endl << endl;

// Subtracting a scalar from an array一个数组减去一个数

cout << "a - 2 = " << endl << a - 2 << endl;

}

//output

a + b =

2 4 6

5 7 9

8 10 12

a - 2 =

-1 0 1

2 3 4

5 6 7

4.数组乘法

数组可以和数值相乘，也可以和数组相乘。但是两个数组相乘与矩阵乘法不一样，数组相乘是对应的系数相乘：

#include <Eigen/Dense>

#include <iostream>

using namespace Eigen;

using namespace std;

int main()

{

ArrayXXf a(2,2);

ArrayXXf b(2,2);

a << 1,2,

3,4;

b << 5,6,

7,8;

cout << "a \* b = " << endl << a \* b << endl;

}

//output

a \* b =

5 12

21 32

5.系数的其他运算

数组类还提供了除了上面加减法、乘法三种运算外的其他运算，比如取绝对值.abs(),开平分根.sqrt(),可以比较两个数组内对应两个元素中的最小值.min(.):

#include <Eigen/Dense>

#include <iostream>

using namespace Eigen;

using namespace std;

int main()

{

ArrayXf a = ArrayXf::Random(5);

a \*= 2;

cout << "a =" << endl

<< a << endl;

cout << "a.abs() =" << endl

<< a.abs() << endl;

cout << "a.abs().sqrt() =" << endl

<< a.abs().sqrt() << endl;

cout << "a.min(a.abs().sqrt()) =" << endl

<< a.min(a.abs().sqrt()) << endl;

}

//output

a =

1.36

-0.422

1.13

1.19

1.65

a.abs() =

1.36

0.422

1.13

1.19

1.65

a.abs().sqrt() =

1.17

0.65

1.06

1.09

1.28

a.min(a.abs().sqrt()) =

1.17

-0.422

1.06

1.09

1.28

6.数组和矩阵之间的转换

你不能在数组类中使用矩阵运算，或者在矩阵类中使用数组运算。因此，当你需要进行线性代数的运算时，使用矩阵类，当你需要对系数进行运算操作时，使用数组类。但是，有时候你既要用到数组类的运算操作，又得用矩阵类的运算操作。在这样的情况下，你可以在数组和矩阵之间进行相互转换。

对矩阵表达而言，可以使用.array()函数来将其转换成数组表达，这样你就可以对系数进行相关运算操作；对于数组表达而言，可以使用.matrix()函数来将其转换为矩阵表达，你也可以进行线性代数的运算了。

在eigen中混用矩阵表达和数组表达是不允许的。比如你不能将数组和矩阵加起来，但是可以转换成同一种表达之后相加。有一种例外情况就是赋值运算符，允许将矩阵表达赋值给数组变量，也允许将数组表达赋值给矩阵变量。

两个数组的乘法等于对于系数相乘，与矩阵函数.cwisseProduct(.)的功能相同：

#include <Eigen/Dense>

#include <iostream>

using namespace Eigen;

using namespace std;

int main()

{

MatrixXf m(2,2);

MatrixXf n(2,2);

MatrixXf result(2,2);

m << 1,2,

3,4;

n << 5,6,

7,8;

result = m \* n;//矩阵乘法

cout << "-- Matrix m\*n: --" << endl << result << endl << endl;

result = m.array() \* n.array();//数组乘法，对应系数相乘

cout << "-- Array m\*n: --" << endl << result << endl << endl;

result = m.cwiseProduct(n);//利用矩阵成员函数对两个矩阵对应系数相乘

cout << "-- With cwiseProduct: --" << endl << result << endl << endl;

result = m.array() + 4;//数组加法，每个元素加4

cout << "-- Array m + 4: --" << endl << result << endl << endl;

}

//output

-- Matrix m\*n: --

19 22

43 50

-- Array m\*n: --

5 12

21 32

-- With cwiseProduct: --

5 12

21 32

-- Array m + 4: --

5 6

7 8

以下是更复杂的运算：

#include <Eigen/Dense>

#include <iostream>

using namespace Eigen;

using namespace std;

int main()

{

MatrixXf m(2,2);

MatrixXf n(2,2);

MatrixXf result(2,2);

m << 1,2,

3,4;

n << 5,6,

7,8;

result = (m.array() + 4).matrix() \* m;//m每个系数先加4再进行矩阵乘法运算

cout << "-- Combination 1: --" << endl << result << endl << endl;

result = (m.array() \* n.array()).matrix() \* m;//m和n的每个对应系数相乘，然后再进行矩阵计算

cout << "-- Combination 2: --" << endl << result << endl << endl;

}