Array类和系数操作

本旨在提供有关如何使用Eigen的Array类的概述和说明。

什么是数组类?

所述Array类提供通用的阵列，而不是 Matrix，其旨在用于线性代数类。此外，Array 类提供了一种简单的方法来执行系数操作，这可能没有线性代数意义，例如向数组中的每个系数添加一个常数或将两个数组按系数相乘。

数组类型

Array是一个类模板，它采用与Matrix相同的模板参数。与Matrix 一样，前三个模板参数是强制性的：

```
Array<typename Scalar, int RowsAtCompileTime, int ColsAtCompileTime>
```

最后三个模板参数是可选的。由于这与Matrix完全相同，我们在此不再赘述，仅参考Matrix 类。

Eigen还为一些常见情况提供了 typedef，其方式类似于Matrix typedef，但有一些细微差别，因为“数组”一词用于一维和二维数组。我们采用这样的约定，即 ArrayNt 形式的 typedef 代表一维数组，其中 N 和 t 是大小和标量类型，如本页中解释的矩阵typedef所示。对于二维数组，我们使用 ArrayNNt 形式的 typedef。下表显示了一些示例：

类型	类型定义
Array<float, Dynamic, 1>	ArrayXf
Array<float, 3, 1>	Array3f
Array<double, Dynamic, Dynamic>	ArrayXXd
Array<double, 3, 3>	Array33d

访问数组中的值

括号运算符被重载以提供对数组系数的读写访问，就像矩阵一样。此外，该 `<<` 运算符可用于初始化数组（通过逗号初始化程序）或打印它们。

例子：

```
#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;
}
```

输出：

```
1 2
3 5

1 2
3 4
```

有关逗号初始化程序的更多信息，请参阅高级初始化。

加减

两个数组的加减与矩阵相同。如果两个数组具有相同的大小，并且加法或减法是按系数进行的，则该操作是有效的。

数组还支持 `array + scalar` 将标量添加到数组中的每个系数的形式的表达式。这提供了不能直接用于Matrix对象的功能。

例子：

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

```

```
    // Subtracting a scalar from an array
    cout << "a - 2 = " << endl << a - 2 << endl;
}
```

输出：

```
a + b =
 2  4  6
 5  7  9
 8 10 12

a - 2 =
-1  0  1
 2  3  4
 5  6  7
```

数组乘法

首先，当然您可以将数组乘以标量，这与矩阵的工作方式相同。数组与矩阵根本不同的地方在于将两个相乘。矩阵将乘法解释为矩阵乘积，数组将乘法解释为系数乘积。因此，两个数组可以相乘当且仅当它们具有相同的维度。

例子：

```
#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;
}
```

输出：

```
a * b =
 5 12
21 32
```

其他系数操作

所述阵列类定义其他系数为单位的运算除了加法，减法和上述乘法运算符。例如，.abs()方法获取每个系数的绝对值，而.sqrt()计算系数的平方根。如果你有两个相同大小的数组，你可以调用.min(.)来构造一个数组，它的系数是两个给定数组对应系数的最小值。以下示例说明了这些操作。

例子：

```cpp
#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;
}
```

输出：

```
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
```

在快速参考指南中可以找到更多的系数操作。

数组和矩阵表达式之间的转换

什么时候应该使用Matrix类的对象，什么时候应该使用Array类的对象？您不能对数组应用矩阵运算，也不能对矩阵应用数组运算。因此，如果您需要进行矩阵乘法等线性代数运算，那么您应该使用矩阵；如果您需要进行系数操作，那么您应该使用数组。但是，有时并不是那么简单，而是需要同时使用矩阵和数组操作。在这种情况下，您需要将矩阵转换为数组或反向转换。无论选择将对象声明为数组还是矩

阵，这都可以访问所有操作。

矩阵表达式有一个.array()方法，可以将它们“转换”为数组表达式，因此可以轻松应用系数操作。相反，数组表达式有一个.matrix()方法。与所有Eigen表达式抽象一样，这没有任何运行时成本（前提是您让编译器优化）。既.array（）和.matrix（）可被用作右值和作为左值。

Eigen禁止在表达式中混合矩阵和数组。例如，您不能直接添加矩阵和数组；运算`+`符的操作数要么都是矩阵，要么都是数组。但是，使用.array()和.matrix()很容易从一种转换到另一种。这条规则的例外是赋值运算符：允许将矩阵表达式赋值给数组变量，或者将数组表达式赋值给矩阵变量。

以下示例显示如何通过使用.array()方法对Matrix对象使用数组操作。例如，该语句采用两个矩阵和，将它们都转换为一个数组，用于将它们按系数相乘并将结果分配给矩阵变量（这是合法的，因为Eigen允许将数组表达式分配给矩阵变量）。`result = m.array() * n.array()`

事实上，这种用法非常普遍，以至于Eigen为矩阵提供了一个const .cwiseProduct(,)方法来计算系数乘积。这也显示在示例程序中。

例子：

```
#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;
    cout << "-- Matrix m*n: --" << endl << result << endl << endl;
    result = m.array() * n.array();
    cout << "-- Array m*n: --" << endl << result << endl << endl;
    result = m.cwiseProduct(n);
    cout << "-- With cwiseProduct: --" << endl << result << endl << endl;
    result = m.array() + 4;
    cout << "-- Array m + 4: --" << endl << result << endl << endl;
}
```

输出：

```

-- 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
```

类似地，如果`array1`和`array2`是数组，则表达式`array1.matrix() * array2.matrix()`计算它们的矩阵乘积。

这是一个更高级的例子。该表达式`(m.array() + 4).matrix() * m`将矩阵中的每个系数加 4，然后计算结果与`m`的矩阵乘积。类似地，表达式`(m.array() * n.array()).matrix() * m`计算矩阵的系数之积`m`和`n`，然后用结果做矩阵积。

例子：

```cpp
#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;
    cout << "-- Combination 1: --" << endl << result << endl << endl;
    result = (m.array() * n.array()).matrix() * m;
    cout << "-- Combination 2: --" << endl << result << endl << endl;
}
```

输出：

```
-- Combination 1: --
23 34
31 46

-- Combination 2: --
 41  58
117 170
```