归约、访问和广播

本页解释了Eigen的归约、访问和广播以及它们如何与矩阵和数组一起使用。

归约

在Eigen 中，**归约是一个函数，它采用矩阵或数组，并返回单个标量值。**最常用的归约之一是.sum()，返回给定矩阵或数组内所有系数的总和。

例子：

```
#include <iostream>
#include <Eigen/Dense>

using namespace std;
int main()
{
    Eigen::Matrix2d mat;
    mat << 1, 2,
        3, 4;
    cout << "Here is mat.sum():       " << mat.sum()       << endl;
    cout << "Here is mat.prod():      " << mat.prod()      << endl;
    cout << "Here is mat.mean():      " << mat.mean()      << endl;
    cout << "Here is mat.minCoeff():  " << mat.minCoeff()  << endl;
    cout << "Here is mat.maxCoeff():  " << mat.maxCoeff()  << endl;
    cout << "Here is mat.trace():     " << mat.trace()     << endl;
}
```

输出：

```
Here is mat.sum():       10
Here is mat.prod():      24
Here is mat.mean():      2.5
Here is mat.minCoeff():  1
Here is mat.maxCoeff():  4
Here is mat.trace():     5
```

函数返回的矩阵的迹`trace()`是对角线系数的总和，可以等效地计算`a.diagonal().sum()`。

范数计算

（欧几里得又名 ℓ^2）可以得到向量的平方范数squaredNorm()。它等于向量自身的点积，等价于其系数的绝对值平方和。

Eigen还提供了norm()方法，该方法返回squaredNorm() 的平方根。

这些操作也可以对矩阵进行操作；在这种情况下，n×p 矩阵被视为大小为 (n*p) 的向量，因此例如norm()方法返回“Frobenius”或“Hilbert-Schmidt”范数。我们避免谈论 ℓ^2 矩阵的范数，因为这可能意味着不同的事情。

如果你想要其他系数方式 ℓ^p 规范，使用lpNorm<p>()方法。如果你想要模板参数*p*可以采用特殊值 ℓ^∞ 范数，它是系数绝对值的最大值。

以下示例演示了这些方法。

例子：

```
#include <Eigen/Dense>
#include <iostream>

using namespace std;
using namespace Eigen;

int main()
{
    VectorXf v(2);
    MatrixXf m(2,2), n(2,2);

    v << -1, 2;
    m << 1, -2, -3, 4;

    cout << "v.squaredNorm() = " << v.squaredNorm() << endl;
    cout << "v.norm() = " << v.norm() << endl;
    cout << "v.lpNorm<1>() = " << v.lpNorm<1>() << endl;
    cout << "v.lpNorm<Infinity>() = " << v.lpNorm<Infinity>() << endl;

    cout << endl;
    cout << "m.squaredNorm() = " << m.squaredNorm() << endl;
    cout << "m.norm() = " << m.norm() << endl;
    cout << "m.lpNorm<1>() = " << m.lpNorm<1>() << endl;
    cout << "m.lpNorm<Infinity>() = " << m.lpNorm<Infinity>() << endl;
}
```

输出：

```
v.squaredNorm() = 5
v.norm() = 2.23607
v.lpNorm<1>() = 3
v.lpNorm<Infinity>() = 2

m.squaredNorm() = 30
m.norm() = 5.47723
m.lpNorm<1>() = 10
m.lpNorm<Infinity>() = 4
```

算子 范数： 1-范数和∞-norm矩阵算子范数可以很容易地计算如下：

例子：

```
#include <Eigen/Dense>
#include <iostream>

using namespace Eigen;
using namespace std;

int main()
{
    MatrixXf m(2,2);
    m << 1,-2,
```

```
        -3,4;

    cout << "1-norm(m)     = " << m.cwiseAbs().colwise().sum().maxCoeff()
        << " == "             << m.colwise().lpNorm<1>().maxCoeff() << endl;

    cout << "infty-norm(m) = " << m.cwiseAbs().rowwise().sum().maxCoeff()
        << " == "             << m.rowwise().lpNorm<1>().maxCoeff() << endl;
}
```

输出：

```
1-norm(m)     = 6 == 6
infty-norm(m) = 7 == 7
```

有关这些表达式的语法的更多解释，请参见下文。

布尔归约

以下归约对布尔值进行操作：

- all()返回**真**，如果所有在给定的系数矩阵或阵列评估为**真**。
- any()返回**真**，如果在给定的系数中的至少一个矩阵或阵列的计算结果为**真**。
- count()返回给定矩阵或数组中计算结果为**true**的系数数。

这些通常与Array提供的按系数比较和相等运算符结合使用。举例来说，`array > 0`是相同的尺寸的阵列`array`，其中**真**在那些位置的相应的系数`array`是正的。因此，检验的所有系数是否为正。这可以在以下示例中看到：`(array > 0).all()`

例子：

```
#include <Eigen/Dense>
#include <iostream>

using namespace std;
using namespace Eigen;

int main()
{
    ArrayXXf a(2,2);

    a << 1,2,
        3,4;

    cout << "(a > 0).all()   = " << (a > 0).all() << endl;
    cout << "(a > 0).any()   = " << (a > 0).any() << endl;
    cout << "(a > 0).count() = " << (a > 0).count() << endl;
    cout << endl;
    cout << "(a > 2).all()   = " << (a > 2).all() << endl;
    cout << "(a > 2).any()   = " << (a > 2).any() << endl;
    cout << "(a > 2).count() = " << (a > 2).count() << endl;
}
```

输出：

```
(a > 0).all()   = 1
(a > 0).any()   = 1
(a > 0).count() = 4

(a > 2).all()   = 0
(a > 2).any()   = 1
(a > 2).count() = 2
```

用户定义的归约

去做

同时，您可以查看 DenseBase::redux() 函数。

访问

当您想要获取Matrix或Array 中系数的位置时，访问很有用。最简单的例子是maxCoeff(&x,&y)和minCoeff(&x,&y)，它们可用于在Matrix或Array 中找到最大或最小系数的位置。

传递给访问者的参数是指向要存储行和列位置的变量的指针。这些变量应该是Index类型，如下所示：

例子：

```cpp
#include <iostream>
#include <Eigen/Dense>

using namespace std;
using namespace Eigen;

int main()
{
    Eigen::MatrixXf m(2,2);

    m << 1, 2,
         3, 4;

    //get location of maximum
    MatrixXf::Index maxRow, maxCol;
    float max = m.maxCoeff(&maxRow, &maxCol);

    //get location of minimum
    MatrixXf::Index minRow, minCol;
    float min = m.minCoeff(&minRow, &minCol);

    cout << "Max: " << max <<  ", at: " <<
        maxRow << "," << maxCol << endl;
    cout << "Min: " << min << ", at: " <<
        minRow << "," << minCol << endl;
}
```

输出：

```
Max: 4, at: 1,1
Min: 1, at: 0,0
```

这两个函数还返回最小或最大系数的值。

部分归约

部分归约是可以在Matrix或Array上按列或按行操作的归约，对每一列或行应用归约操作并返回具有相应值的列或行向量。使用colwise()或rowwise()应用部分减少。

一个简单的例子是获取给定矩阵中每列元素的最大值，并将结果存储在行向量中：

例子：

```
#include <iostream>
#include <Eigen/Dense>

using namespace std;
int main()
{
    Eigen::MatrixXf mat(2,4);
    mat << 1, 2, 6, 9,
        3, 1, 7, 2;

    std::cout << "Column's maximum: " << std::endl
        << mat.colwise().maxCoeff() << std::endl;
}
```

输出：

```
Column's maximum:
3 2 7 9
```

可以按行执行相同的操作：

例子：

```
#include <iostream>
#include <Eigen/Dense>

using namespace std;
int main()
{
    Eigen::MatrixXf mat(2,4);
    mat << 1, 2, 6, 9,
        3, 1, 7, 2;

    std::cout << "Row's maximum: " << std::endl
        << mat.rowwise().maxCoeff() << std::endl;
}
```

输出：

```
Row's maximum:
9
7
```

请注意，逐列操作返回一个行向量，而逐行操作返回一个列向量。

将部分归约与其他操作相结合

也可以使用部分归约的结果做进一步处理。这是另一个示例，用于查找矩阵中元素总和为最大值的列。通过逐列部分减少，这可以编码为：

例子：

```
#include <iostream>
#include <Eigen/Dense>

using namespace std;
using namespace Eigen;
int main()
{
    MatrixXf mat(2,4);
    mat << 1, 2, 6, 9,
        3, 1, 7, 2;

    MatrixXf::Index   maxIndex;
    float maxNorm = mat.colwise().sum().maxCoeff(&maxIndex);

    std::cout << "Maximum sum at position " << maxIndex << std::endl;

    std::cout << "The corresponding vector is: " << std::endl;
    std::cout << mat.col( maxIndex ) << std::endl;
    std::cout << "And its sum is is: " << maxNorm << std::endl;
}
```

输出：

```
Maximum sum at position 2
The corresponding vector is:
6
7
And its sum is is: 13
```

前面的示例通过colwise()访问者对每列应用sum()约简，从而获得大小为 1x4 的新矩阵。

因此，如果

$$m = \begin{bmatrix} 1 & 2 & 6 & 9 \\ 3 & 1 & 7 & 2 \end{bmatrix} \tag{1}$$

然后

$$m.colwise().sum() = \begin{bmatrix} 4 & 3 & 13 & 11 \end{bmatrix} \tag{2}$$

所述maxCoeff()减少，最后施加到获得其中最大总和被发现的列索引，这是在这种情况下，列索引 2（第三列）。

广播

广播背后的概念类似于部分归约，不同之处在于广播构建了一个表达式，其中通过在一个方向上复制向量（列或行）被解释为矩阵。

一个简单的例子是将某个列向量添加到矩阵中的每一列。这可以通过以下方式完成：

例子：

```
#include <iostream>
#include <Eigen/Dense>

using namespace std;
int main()
{
    Eigen::MatrixXf mat(2,4);
    Eigen::VectorXf v(2);

    mat << 1, 2, 6, 9,
        3, 1, 7, 2;

    v << 0, 1;

    //add v to each column of m
    mat.colwise() += v;

    std::cout << "Broadcasting result: " << std::endl;
    std::cout << mat << std::endl;
}
```

输出：

```
Broadcasting result:
1 2 6 9
4 2 8 3
```

我们可以`mat.colwise() += v`用两种等效的方式解释指令。它将向量添加`v`到矩阵的每一列。或者，它可以解释为将向量重复`v`四次以形成一个四乘二的矩阵，然后将其添加到`mat:`

$$\begin{bmatrix} 1 & 2 & 6 & 9 \\ 3 & 1 & 7 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 6 & 9 \\ 4 & 2 & 8 & 3 \end{bmatrix} \tag{3}$$

运用`-=`，`+`并且`-`也可用于逐列和行操作。在阵列上，我们也可以使用运算符`*=`，`/=`，`*`和`/`执行系数逐乘除逐列或逐行。这些运算符在矩阵上不可用，因为不清楚它们会做什么。如果要将矩阵的第 0 列与相乘，将第 1 列`mat`与相乘，依此类推，请使用。`mat = mat * v.asDiagonal()`

需要指出的是，要按列或按行添加的向量必须是 Vector 类型，不能是Matrix。如果不满足，则会出现编译时错误。这也意味着当使用Matrix操作时，广播操作只能应用于 Vector 类型的对象。这同样适用于Array类，其中 VectorXf 的等效项是 ArrayXf。与往常一样，您不应在同一表达式中混合使用数组和矩阵。

要按行执行相同的操作，我们可以执行以下操作：

例子：

```
#include <iostream>
#include <Eigen/Dense>

using namespace std;
int main()
{
    Eigen::MatrixXf mat(2,4);
    Eigen::VectorXf v(4);
```

```
    mat << 1, 2, 6, 9,
        3, 1, 7, 2;

    v << 0,1,2,3;

    //add v to each row of m
    mat.rowwise() += v.transpose();

    std::cout << "Broadcasting result: " << std::endl;
    std::cout << mat << std::endl;
}
```

输出：

```
Broadcasting result:
 1  3  8 12
 3  2  9  5
```

将广播与其他操作相结合

广播还可以与其他操作相结合，例如矩阵或数组操作、归约和部分归约。

现在已经引入了广播、减少和部分减少，我们可以深入研究一个更高级的例子，它`v`在矩阵的列中找到向量的最近邻居`m`。本例中将使用欧几里德距离，使用名为squaredNorm()的部分归约计算平方欧几里德距离：

例子：

```
#include <iostream>
#include <Eigen/Dense>

using namespace std;
using namespace Eigen;

int main()
{
    Eigen::MatrixXf m(2,4);
    Eigen::VectorXf v(2);

    m << 1, 23, 6, 9, 3, 11, 7, 2;

    v << 2, 3;

    MatrixXf::Index index;
    // find nearest neighbour
    (m.colwise() - v).colwise().squaredNorm().minCoeff(&index);

    cout << "Nearest neighbour is column " << index << ":" << endl;
    cout << m.col(index) << endl;
}
```

输出：

```
Nearest neighbour is column 0:
1
3
```

完成这项工作的线路是

`(m.colwise() - v).colwise().squaredNorm().minCoeff(&index);`

我们将逐步了解正在发生的事情：

- m.colwise() - v 是广播操作，此操作的结果是一个大小与 matrix 相同的新矩阵 m

$$m.colwise() - v = \begin{bmatrix} -1 & 21 & 4 & 7 \\ 0 & 8 & 4 & -1 \end{bmatrix} \tag{4}$$

- `(m.colwise() - v).colwise().squaredNorm()` 是部分归约，按列计算平方范数。此操作的结果是一个行向量，其中每个系数是 m 和 v 中每列之间的平方欧几里得距离：

$$(m.colwise() - v).colwise().squaredNorm() = \begin{bmatrix} 1 & 505 & 32 & 50 \end{bmatrix} \tag{5}$$

- 最后，`minCoeff(&index)` 用于获得 `m` 以 `v` 欧几里德距离而言最接近的列的索引。