切片和索引

密集矩阵和数组操作

本页介绍了`operator()`索引行和列子集所提供的众多可能性。此 API 已在 Eigen 3.4 中引入。它支持块 API提出的所有功能，等等。特别是，它支持**切片**，包括采用一组行、列或元素，在矩阵内均匀间隔或从索引数组索引。

概述

所有上述操作都是通过通用的`DenseBase::operator()(const RowIndices&, const ColIndices&)`方法处理的。每个参数可以是：

- 索引单个行或列的整数，包括符号索引。
- 符号Eigen::all以升序表示整个行或列的集合。
- 由Eigen::seq、Eigen::seqN或Eigen::lastN函数构造的ArithmeticSequence。
- 任何一维向量/整数数组，包括 Eigen 的向量/数组、表达式、std::vector、std::array 以及普通的 C 数组：`int[N]`。

更一般地说，它可以接受任何公开以下两个成员函数的对象：

```
<integral type> operator[](<integral type>) const;
<integral type> size() const;
```

其中`<integral type>`代表与Eigen::Index兼容的任何整数类型（即`std::ptrdiff_t`）。

基本切片

通过Eigen::seq或Eigen::seqN函数获取矩阵或向量内均匀间隔的一组行、列或元素，其中“seq”代表算术序列。他们的签名总结如下：

功能	描述	例子
seq (firstIdx,lastIdx)	表示范围从`firstIdx`到的整数序列`lastIdx`	seq(2,5) <=> {2,3,4,5}
seq (firstIdx,lastIdx,incr)	相同但使用增量`incr`从一个索引前进到下一个	seq（2,8,2）<=> {2,4,6,8}
seqN (firstIdx,size)	表示`size`整数序列，从`firstIdx`	seqN (2,5) <=> {2,3,4,5,6}
seqN (firstIdx,size,incr)	相同但使用增量`incr`从一个索引前进到下一个	seqN (2,3,3) <=> {2,5,8}

`firstIdx`和`lastIdx`参数也可以与所述的帮助下所定义 Eigen::last 代表最后行，列或底层矩阵/向量的元素的索引一次运算序列通过操作员传递给它的符号。以下是 2D array/matrix`A`和 1D array/vector的一些示例`v`。

意图	代码	块 API 等价
从`i`带有`n`列的行开始的左下角	A(seq (i, last), seqN (0, n))	A.bottomLeftCorner(A.rows() - i, n)
从`i`,j开始的块有`m`行和`n`列	A(seqN (i, m), seqN (i, n)	A.block(i, j, m, n)
块开始`i0`，`j0`和结束`i1`，`j1`	A（seq（i0，i1），seq（j0，j1）	A.block(i0, j0, i1 - i0 + 1,j1 - j0 + 1)
偶数列 A	A(all, seq (0, last ,2))	
奇数行 A	A(seqN (1,n,2), all)	
倒数第二列	A(all,last-1)	A.col(A.cols() - 2)
中间一行	A(last/2，all)	A.row((A.rows() - 1) / 2)
v 从 i 开始的最后一个元素	v(seq (i, last))	v.tail(v.size() - i)
v 的最后`n`个元素	v(seq (last +1-n, last))	v.tail(n)

如上一个示例所示，引用*最后 n 个*元素（或行/列）编写起来有点麻烦。对于非默认增量，这变得更加棘手且容易出错。这里是Eigen::lastN(size)和Eigen::lastN(size,incr)：

意图	代码	块 API 等价
v 的最后`n`个元素	v(lastN (n))	v.tail(n)
A的右下角的大小`m`*`n`	v(lastN (m), lastN (n))	A.bottomRightCorner(m,n)
最后一`n`列占 1 列超过 3	A(all, lastN (n,3))	

编译时间大小和增量

在性能方面，Eigen 和编译器可以利用编译时大小和增量。为此，您可以使用Eigen::fix强制执行编译时参数。这样的编译时值可以与Eigen::last符号结合使用：

v(seq (last -fix<7>, last -fix<2>))

在这个例子中，Eigen 在编译时知道返回的表达式有 6 个元素。它相当于：

v(seqN (last -7, fix<6>))

我们可以重新访问*A示例的偶数列，*如下所示：

A(all , seq (0, last ,fix<2>))

相反的顺序

也可以使用负增量以降序枚举行/列索引。例如，从第 20 列到第 10 列的 A 列多于两列：

A(all , seq (20, 10, fix<-2>))

从最后`n`一行开始的最后一行：

A(seqN (last , n, fix<-1>), all)

您还可以使用 ArithmeticSequence::reverse() 方法来反转其顺序。因此，前面的例子也可以写成：

A(lastN (n).reverse(), all)

索引数组

通用`operator()`也作为输入存储为一个行或列指数的任意列表`ArrayXi`，一个`std::vector<int>`，`std::array<int,N>`等等。

例子：

```
std::vector<int> ind{4,2,5,5,3};
MatrixXi A = MatrixXi::Random(4,6);
cout << "Initial matrix A:\n" << A << "\n\n";
cout << "A(all,ind):\n" << A(all,ind) << "\n\n";
```

输出：

```
Initial matrix A:
  7   9  -5  -3   3 -10
 -2  -6   1   0   5  -5
  6  -3   0   9  -8  -8
  6   6   3   9   2   6

A(all,ind):
  3  -5 -10 -10  -3
  5   1  -5  -5   0
 -8   0  -8  -8   9
  2   3   6   6   9
```

你也可以直接传递一个静态数组：

例子：

```
#if EIGEN_HAS_STATIC_ARRAY_TEMPLATE
MatrixXi A = MatrixXi::Random(4,6);
cout << "Initial matrix A:\n" << A << "\n\n";
cout << "A(all,{4,2,5,5,3}):\n" << A(all,{4,2,5,5,3}) << "\n\n";
#endif
```

输出：

```
Initial matrix A:
  7   9  -5  -3   3 -10
 -2  -6   1   0   5  -5
  6  -3   0   9  -8  -8
  6   6   3   9   2   6

A(all,{4,2,5,5,3}):
  3  -5 -10 -10  -3
  5   1  -5  -5   0
 -8   0  -8  -8   9
  2   3   6   6   9
```

或表达式：

例子：

```
ArrayXi ind(5); ind<<4,2,5,5,3;
MatrixXi A = MatrixXi::Random(4,6);
cout << "Initial matrix A:\n" << A << "\n\n";
cout << "A(all,ind-1):\n" << A(all,ind-1) << "\n\n";
```

输出：

```
Initial matrix A:
  7   9  -5  -3   3 -10
 -2  -6   1   0   5  -5
  6  -3   0   9  -8  -8
  6   6   3   9   2   6

A(all,ind-1):
-3  9  3  3 -5
 0 -6  5  5  1
 9 -3 -8 -8  0
 9  6  2  2  3
```

当与编译时大小传递对象如`Array4i`，`std::array<int,N>`，或一个静态数组，则返回的表达也显示出编译时的尺寸。

自定义索引列表

更一般地，`operator()`可以接受与以下`ind`类型`T`兼容的任何对象作为输入：

```
Index s = ind.size(); or Index s = size(ind);
Index i;
i = ind[i];
```

这意味着您可以轻松构建自己的花哨序列生成器并将其传递给`operator()`. 这是一个放大给定矩阵的示例，同时通过重复填充额外的第一行和列：

例子：

```
struct pad {
  Index size() const { return out_size; }
  Index operator[] (Index i) const { return std::max<Index>(0,i-(out_size-
in_size)); }
  Index in_size, out_size;
};

Matrix3i A;
A.reshaped() = VectorXi::LinSpaced(9,1,9);
cout << "Initial matrix A:\n" << A << "\n\n";
MatrixXi B(5,5);
B = A(pad{3,5}, pad{3,5});
cout << "A(pad{3,N}, pad{3,N}):\n" << B << "\n\n";
```

输出：

```
Initial matrix A:
1 4 7
2 5 8
3 6 9

A(pad{3,N}, pad{3,N}):
1 1 1 4 7
1 1 1 4 7
1 1 1 4 7
2 2 2 5 8
3 3 3 6 9
```