1.4 块运算

块是矩阵或数组的一个矩形部分。块表达式既可以做左值也可以作右值。和矩阵表达式一样，块分解具有零运行时间成本，对你的程序进行优化。

1.使用块运算

最常用的块运算是.block()成员函数。以下是两个版本的块定义：

|  |  |  |
| --- | --- | --- |
| 块运算 | 动态大小的块定义版本 | 指定大小的块定义版本 |
| 定义从第i行第j列开始的大小为PxQ的块 | matrix.block(i,j,p,q); | matrix.block<p,q>(i,j); |

**在eigen中行数和列数都是从0开始的！**

这两个版本可以用于指定大小和动态大小的矩阵和数组类。这两种表达语法上是一样的，唯一不同的是对于小尺寸块而言，指定大小的版本能显著地提供代码运算能力，但是需要在编译时知道其大小。

#include <Eigen/Dense>

#include <iostream>

using namespace std;

int main()

{

Eigen::MatrixXf m(4,4);

m << 1, 2, 3, 4,

5, 6, 7, 8,

9,10,11,12,

13,14,15,16;

cout << "Block in the middle" << endl;

cout << m.block<2,2>(1,1) << endl << endl;

for (int i = 1; i <= 3; ++i)

{

cout << "Block of size " << i << "x" << i << endl;

cout << m.block(0,0,i,i) << endl << endl;

}

}

//output

Block in the middle

6 7

10 11

Block of size 1x1

1

Block of size 2x2

1 2

5 6

Block of size 3x3

1 2 3

5 6 7

9 10 11

在上面的例子中，块函数被看作了右值，即它仅读数据。但是块也可以作为左值，也就是说你可以对它进行赋值：

#include <Eigen/Dense>

#include <iostream>

using namespace std;

using namespace Eigen;

int main()

{

Array22f m;

m << 1,2,

3,4;

Array44f a = Array44f::Constant(0.6);

cout << "Here is the array a:" << endl << a << endl << endl;

a.block<2,2>(1,1) = m;

cout << "Here is now a with m copied into its central 2x2 block:" << endl << a << endl << endl;

a.block(0,0,2,3) = a.block(2,1,2,3);

cout << "Here is now a with bottom-right 2x3 block copied into top-left 2x2 block:" << endl << a << endl << endl;

}

//output

Here is the array a:

0.6 0.6 0.6 0.6

0.6 0.6 0.6 0.6

0.6 0.6 0.6 0.6

0.6 0.6 0.6 0.6

Here is now a with m copied into its central 2x2 block:

0.6 0.6 0.6 0.6

0.6 1 2 0.6

0.6 3 4 0.6

0.6 0.6 0.6 0.6

Here is now a with bottom-right 2x3 block copied into top-left 2x2 block:

3 4 0.6 0.6

0.6 0.6 0.6 0.6

0.6 3 4 0.6

0.6 0.6 0.6 0.6

2.列和行

单列和单行是块的特例。eigen提供更简单的方法获得单行和单列：分别是.row()和.col()

以下为操作语法：

|  |  |
| --- | --- |
| Block operation | Method |
| 第i行 | matrix.row(i); |
| 第j列 | matrix.col(j); |

这里的行列数与eigen矩阵内部的行列数相同，都是从0开始的。

#include <Eigen/Dense>

#include <iostream>

using namespace std;

int main()

{

Eigen::MatrixXf m(3,3);

m << 1,2,3,

4,5,6,

7,8,9;

cout << "Here is the matrix m:" << endl << m << endl;

cout << "2nd Row: " << m.row(1) << endl;

m.col(2) += 3 \* m.col(0);

cout << "After adding 3 times the first column into the third column, the matrix m is:\n";

cout << m << endl;

}

//output

Here is the matrix m:

1 2 3

4 5 6

7 8 9

2nd Row: 4 5 6

After adding 3 times the first column into the third column, the matrix m is:

1 2 6

4 5 18

7 8 30

3.对角相关运算

Eigen还提供了针对矩阵或数组的某个角或边齐平的块的特殊方法。比如,成员函数.topLeftCorner()表示块的左上角的矩阵。

以下是不同方向的对角阵的表示方法：

|  |  |  |
| --- | --- | --- |
| Block **operation** | 动态大小版本 | 指定大小版本 |
| 从左上角数包含p行q列的块 | matrix.topLeftCorner(p,q); | matrix.topLeftCorner<p,q>(); |
| 从左下角数包含p行q列的块 | matrix.bottomLeftCorner(p,q); | matrix.bottomLeftCorner<p,q>(); |
| 从右上角数包含p行q列的块 | matrix.topRightCorner(p,q); | matrix.topRightCorner<p,q>(); |
| 从右下角数包含p行q列的块 | matrix.bottomRightCorner(p,q); | matrix.bottomRightCorner<p,q>(); |
| 包含前q行的块 | matrix.topRows(q); | matrix.topRows<q>(); |
| 包含后q行的块 | matrix.bottomRows(q); | matrix.bottomRows<q>(); |
| 包含前p列的块 | matrix.leftCols(p); | matrix.leftCols<p>(); |
| 包含后p列的块 | matrix.rightCols(q); | matrix.rightCols<q>(); |

以下为实例：

#include <Eigen/Dense>

#include <iostream>

using namespace std;

int main()

{

Eigen::Matrix4f m;

m << 1, 2, 3, 4,

5, 6, 7, 8,

9, 10,11,12,

13,14,15,16;

cout << "m.leftCols(2) =" << endl << m.leftCols(2) << endl << endl;//取左边两列

cout << "m.bottomRows<2>() =" << endl << m.bottomRows<2>() << endl << endl;//取底部两行

m.topLeftCorner(1,3) = m.bottomRightCorner(3,1).transpose();

cout << "After assignment, m = " << endl << m << endl;

}

//output

m.leftCols(2) =

1 2

5 6

9 10

13 14

m.bottomRows<2>() =

9 10 11 12

13 14 15 16

After assignment, m =

8 12 16 4

5 6 7 8

9 10 11 12

13 14 15 16

4.向量的块运算

eigen提供给向量和一维数组一系列的特定块操作：

|  |  |  |
| --- | --- | --- |
| 块运算 | 动态大小版本 | 指定大小版本 |
| 包含前n个元素的块 | vector.head(n); | vector.head<n>(); |
| 包含后n个元素的块 | vector.tail(n); | vector.tail<n>(); |
| 包含从第i个元素开始后n个元素的块 | vector.segment(i,n); | vector.segment<n>(i); |

以下是使用实例：

#include <Eigen/Dense>

#include <iostream>

using namespace std;

int main()

{

Eigen::ArrayXf v(6);

v << 1, 2, 3, 4, 5, 6;

cout << "v.head(3) =" << endl << v.head(3) << endl << endl;

cout << "v.tail<3>() = " << endl << v.tail<3>() << endl << endl;

v.segment(1,4) \*= 2;

cout << "after 'v.segment(1,4) \*= 2', v =" << endl << v << endl;

}

//output

v.head(3) =

1

2

3

v.tail<3>() =

4

5

6

after 'v.segment(1,4) \*= 2', v =

1

4

6

8

10

6