高级初始化

密集矩阵和数组操作

本页讨论了几种用于初始化矩阵的高级方法。它提供了之前介绍的逗号初始化程序的更多详细信息。它还解释了如何获得特殊矩阵，例如单位矩阵和零矩阵。

逗号初始化器

Eigen提供了一个逗号初始化语法，允许用户轻松设置矩阵、向量或数组的所有系数。简单地列出系数，从左上角开始，从左到右，从上到下。需要事先指定对象的大小。如果您列出的系数太少或太多，Eigen会抱怨。

例子：

```
Matrix3f m;
m << 1, 2, 3,
     4, 5, 6,
     7, 8, 9;
std::cout << m;
```

输出：

```
1 2 3
4 5 6
7 8 9
```

此外，初始化列表的元素本身可以是向量或矩阵。一个常见的用途是将向量或矩阵连接在一起。例如，这里是如何将两个行向量连接在一起。请记住，您必须先设置大小，然后才能使用逗号初始值设定项。

例子：

```
RowVectorXd vec1(3);
vec1 << 1, 2, 3;
std::cout << "vec1 = " << vec1 << std::endl;

RowVectorXd vec2(4);
vec2 << 1, 4, 9, 16;
std::cout << "vec2 = " << vec2 << std::endl;

RowVectorXd joined(7);
joined << vec1, vec2;
std::cout << "joined = " << joined << std::endl;
```

输出：

```
vec1 = 1 2 3
vec2 =  1  4  9 16
joined =  1  2  3  1  4  9 16
```

我们可以使用相同的技术来初始化具有块结构的矩阵。

例子：

```
MatrixXf matA(2, 2);
matA << 1, 2, 3, 4;
MatrixXf matB(4, 4);
matB << matA, matA/10, matA/10, matA;
std::cout << matB << std::endl;
```

输出：

```
  1   2 0.1 0.2
  3   4 0.3 0.4
0.1 0.2   1   2
0.3 0.4   3   4
```

逗号初始值设定项也可用于填充块表达式，例如`m.row(i)`. 这是获得与上面第一个示例相同的结果的更复杂的方法：

例子：

```
Matrix3f m;
m.row(0) << 1, 2, 3;
m.block(1,0,2,2) << 4, 5, 7, 8;
m.col(2).tail(2) << 6, 9;
std::cout << m;
```

输出：

```
1 2 3
4 5 6
7 8 9
```

特殊矩阵和数组

矩阵和阵列类具有静态方法等Zero()，其可用于所有系数初始化到零。共有三种变体。第一个变体没有参数，只能用于固定大小的对象。如果要将动态大小对象初始化为零，则需要指定大小。因此，第二个变体需要一个参数并且可以用于一维动态大小的对象，而第三个变体需要两个参数并且可以用于二维对象。以下示例说明了所有三种变体：

例子：

```
std::cout << "A fixed-size array:\n";
Array33f a1 = Array33f::Zero();
std::cout << a1 << "\n\n";


std::cout << "A one-dimensional dynamic-size array:\n";
ArrayXf a2 = ArrayXf::Zero(3);
std::cout << a2 << "\n\n";


std::cout << "A two-dimensional dynamic-size array:\n";
ArrayXXf a3 = ArrayXXf::Zero(3, 4);
std::cout << a3 << "\n";
```

输出：

```
A fixed-size array:
0 0 0
0 0 0
0 0 0

A one-dimensional dynamic-size array:
0
0
0

A two-dimensional dynamic-size array:
0 0 0 0
0 0 0 0
0 0 0 0
```

同样，静态方法Constant (value) 将所有系数设置为`value`。如果需要指定对象的大小，则附加参数位于`value`参数之前，如`MatrixXd::Constant(rows, cols, value)`. Random()方法用随机系数填充矩阵或数组。可以通过调用Identity()获得单位矩阵；此方法仅适用于Matrix，不适用于Array，因为“单位矩阵”是一个线性代数概念。该方法LinSpaced（尺寸，低，高）是仅可用于载体和一维数组; 它产生一个指定大小的向量，其系数在`low`和之间等距`high`。方法`LinSpaced()`如下例所示，该示例打印了一个表，其中包含以度为单位的角度、以弧度为单位的相应角度以及它们的正弦和余弦。

例子：

```
ArrayXXf table(10, 4);
table.col(0) = ArrayXf::LinSpaced(10, 0, 90);
table.col(1) = M_PI / 180 * table.col(0);
table.col(2) = table.col(1).sin();
table.col(3) = table.col(1).cos();
std::cout << "  Degrees   Radians      Sine    Cosine\n";
std::cout << table << std::endl;
```

输出：

```
  Degrees   Radians      Sine    Cosine
        0         0         0         1
       10     0.175     0.174     0.985
       20     0.349     0.342      0.94
       30     0.524       0.5     0.866
       40     0.698     0.643     0.766
       50     0.873     0.766     0.643
       60      1.05     0.866       0.5
       70      1.22      0.94     0.342
       80       1.4     0.985     0.174
       90      1.57         1 -4.37e-08
```

这个例子表明，像 LinSpaced() 返回的对象一样，可以分配给变量（和表达式）。Eigen定义了实用函数，如setZero()、MatrixBase::setIdentity()和DenseBase::setLinSpaced()以方便地执行此操作。下面的例子对比了三种构造矩阵方法：使用静态方法和赋值，使用静态方法和逗号 -初始化程序，或使用 setXxx() 方法。

例子：

```
const int size = 6;
MatrixXd mat1(size, size);
mat1.topLeftCorner(size/2, size/2)     = MatrixXd::Zero(size/2, size/2);
mat1.topRightCorner(size/2, size/2)    = MatrixXd::Identity(size/2, size/2);
mat1.bottomLeftCorner(size/2, size/2)  = MatrixXd::Identity(size/2, size/2);
mat1.bottomRightCorner(size/2, size/2) = MatrixXd::Zero(size/2, size/2);
std::cout << mat1 << std::endl << std::endl;

MatrixXd mat2(size, size);
mat2.topLeftCorner(size/2, size/2).setZero();
mat2.topRightCorner(size/2, size/2).setIdentity();
mat2.bottomLeftCorner(size/2, size/2).setIdentity();
mat2.bottomRightCorner(size/2, size/2).setZero();
std::cout << mat2 << std::endl << std::endl;

MatrixXd mat3(size, size);
mat3 << MatrixXd::Zero(size/2, size/2), MatrixXd::Identity(size/2, size/2),
        MatrixXd::Identity(size/2, size/2), MatrixXd::Zero(size/2, size/2);
std::cout << mat3 << std::endl;
```

输出：

```
0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 0 1
1 0 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0

0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 0 1
1 0 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0

0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 0 1
1 0 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0
```

可以在快速参考指南 中找到所有预定义矩阵、向量和数组对象的摘要。

用作临时对象

如上所示，Zero() 和 Constant() 等静态方法可用于在声明时或赋值运算符的右侧初始化变量。您可以将这些方法视为返回矩阵或数组；事实上，它们返回所谓的表达式对象，这些对象在需要时计算为矩阵或数组，因此这种语法不会产生任何开销。

这些表达式也可以用作临时对象。我们在此处复制的入门指南中的第二个示例已经说明了这一点。

例子：

```
#include <iostream>
```

```cpp
#include <Eigen/Dense>

using namespace Eigen;
using namespace std;

int main()
{
    MatrixXd m = MatrixXd::Random(3,3);
    m = (m + MatrixXd::Constant(3,3,1.2)) * 50;
    cout << "m =" << endl << m << endl;
    VectorXd v(3);
    v << 1, 2, 3;
    cout << "m * v =" << endl << m * v << endl;
}
```

输出：

```
m =
  94 89.8 43.5
49.4  101 86.8
88.3 29.8 37.8
m * v =
404
512
261
```

该表达式构造了 3×3 矩阵表达式，其所有系数都等于 1.2 加上相应的*m*系数。`m + MatrixXf::Constant(3,3,1.2)`

逗号初始化器也可用于构造临时对象。下面的示例构造一个大小为 2×3 的随机矩阵。

例子：

```cpp
MatrixXf mat = MatrixXf::Random(2, 3);
std::cout << mat << std::endl << std::endl;
mat = (MatrixXf(2,2) << 0, 1, 1, 0).finished() * mat;
std::cout << mat << std::endl;
```

输出：

```
  0.68  0.566  0.823
-0.211  0.597 -0.605

-0.211  0.597 -0.605
  0.68  0.566  0.823
```

在这里，一旦临时矩阵的逗号初始化完成，利用 finished()方法来获得实际的矩阵对象。