原地矩阵分解

密集线性问题和分解

从 Eigen 3.3 开始，LU、Cholesky 和 QR 分解可以*就地*操作，即直接在给定的输入矩阵内操作。当处理巨大的矩阵时，或当可用内存非常有限（嵌入式系统）时，此功能特别有用。

为此，必须使用 Ref<> 矩阵类型实例化相应的分解类，并且必须使用输入矩阵作为参数构造分解对象。作为一个例子，让我们考虑一个带有部分旋转的就地 LU 分解。

让我们从基本的包含和 2x2 矩阵 `A` 的声明开始：

代码：

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

using namespace std;
using namespace Eigen;

int main()
{
    MatrixXd A(2,2);
    A << 2, -1, 1, 3;
    cout << "Here is the input matrix A before decomposition:\n" << A <<
endl;
}
```

输出：

```
Here is the input matrix A before decomposition:
 2 -1
 1  3
```

这里没有惊喜！然后，让我们声明我们的就地 LU 对象 `lu`，并检查矩阵 `A` 的内容：

```
PartialPivLU<Ref<MatrixXd> > lu(A);
cout << "Here is the input matrix A after decomposition:\n" << A << endl;
```

```
Here is the input matrix A after decomposition:
  2  -1
0.5 3.5
```

这里，lu对象计算L和U因子，并将其存储在矩阵A所持有的内存中。因此，A的系数在因子分解过程中被破坏，并被L和U因子替换，正如人们可以验证的那样：

```
cout << "Here is the matrix storing the L and U factors:\n" << lu.matrixLU()
<< endl;
```

```
Here is the matrix storing the L and U factors:
  2  -1
0.5 3.5
```

然后，可以`lu`像往常一样使用该对象，例如解决 Ax=b 问题：

```
MatrixXd A0(2,2); A0 << 2, -1, 1, 3;
VectorXd b(2);    b << 1, 2;
VectorXd x = lu.solve(b);
cout << "Residual: " << (A0 * x - b).norm() << endl;
```

```
Residual: 0
```

在这里，由于原始矩阵的内容`A`已经丢失，我们不得不声明一个新矩阵`A0`来验证结果。

由于内存在`A`和之间共享`lu`，因此修改矩阵`A`将使`lu`无效。这可以通过修改内容`A`并再次尝试解决初始问题来轻松验证：

```
A << 3, 4, -2, 1;
x = lu.solve(b);
cout << "Residual: " << (A0 * x - b).norm() << endl;
```

```
Residual: 15.8114
```

请注意，引擎盖下没有共享指针，只要lu还活着，用户就有责任在生命中保留输入矩阵A。

如果要使用修改后的 A 更新因式分解，则必须像往常一样调用计算方法：

```
A0 = A; // save A
lu.compute(A);
x = lu.solve(b);
cout << "Residual: " << (A0 * x - b).norm() << endl;
```

```
Residual: 0
```

请注意，调用compute不会更改lu对象引用的内存。因此，如果使用与A不同的另一个矩阵A1调用计算方法，则不会修改A1的内容。这仍然是将用于存储矩阵A1的L和U因子的A的内容。这很容易通过以下方式进行验证：

```
MatrixXd A1(2,2);
A1 << 5,-2,3,4;
lu.compute(A1);
cout << "Here is the input matrix A1 after decomposition:\n" << A1 << endl;
```

```
Here is the input matrix A1 after decomposition:
 5 -2
 3  4
```

矩阵`A1`不变，这样就可以求解A1*x=b，直接检查残差，无需复制`A1:`

```
x = lu.solve(b);
cout << "Residual: " << (A1 * x - b).norm() << endl;
```

```
Residual: 2.48253e-16
```

以下是支持这种就地机制的矩阵分解列表：

- class LLT
- class LDLT
- class PartialPivLU
- class FullPivLU
- class HouseholderQR
- class ColPivHouseholderQR
- class FullPivHouseholderQR
- class CompleteOrthogonalDecomposition