1.2 矩阵和向量的运算

1.介绍

eigen给矩阵和向量的算术运算提供重载的c++算术运算符例如+，-，\*或这一些点乘dot(),叉乘cross()等等。对于矩阵类（矩阵和向量，之后统称为矩阵类），算术运算只重载线性代数的运算。例如matrix1\*matrix2表示矩阵的乘法，同时向量+标量是不允许的！如果你想进行所有的数组算术运算，请看下一节！

2.加减法

因为eigen库无法自动进行类型转换，因此矩阵类的加减法必须是两个同类型同维度的矩阵类相加减。

这些运算有：

双目运算符：+，a+b

双目运算符：-，a-b

单目运算符：-，-a

复合运算符：+=，a+=b

复合运算符：-=，a-=b

例子：

#include <iostream>

#include <Eigen/Dense>

using namespace Eigen;

int main()

{

Matrix2d a;

a << 1, 2,

3, 4;

MatrixXd b(2,2);

b << 2, 3,

1, 4;

std::cout << "a + b =\n" << a + b << std::endl;

std::cout << "a - b =\n" << a - b << std::endl;

std::cout << "Doing a += b;" << std::endl;

a += b;

std::cout << "Now a =\n" << a << std::endl;

Vector3d v(1,2,3);

Vector3d w(1,0,0);

std::cout << "-v + w - v =\n" << -v + w - v << std::endl;

}

3.标量乘法和除法

标量的乘除法非常简单：

双目运算符：\*，matrix\*scalar

双目运算符：\*，scalar\*matrix

即乘法满足交换律

双目运算符：/，matrix/scalar

矩阵中的每一个元素除以标量

复合运算符：\*=，matrix\*=scalar

复合运算符：/=，matrix/=scalar

Output:

#include <iostream>

#include <Eigen/Dense>

using namespace Eigen;

int main()

{

Matrix2d a;

a << 1, 2,

3, 4;

Vector3d v(1,2,3);

std::cout << "a \* 2.5 =\n" << a \* 2.5 << std::endl;

std::cout << "0.1 \* v =\n" << 0.1 \* v << std::endl;

std::cout << "Doing v \*= 2;" << std::endl;

v \*= 2;

std::cout << "Now v =\n" << v << std::endl;

}

//output

a \* 2.5 =

2.5 5

7.5 10

0.1 \* v =

0.1

0.2

0.3

Doing v \*= 2;

Now v =

2

4

6

4.对表达式模板的注释

在eigen中，+号算术运算符不会通过自身函数执行任何计算，它们只是返回一个表达式，来描述计算的过程。实际的计算是在执行等号时，整个表达式开始进行计算。

比如：

VectorXf a(50), b(50), c(50), d(50);

...

a = 3\*b + 4\*c + 5\*d;

eigen把它编译成一个循环，这样数组只执行依次运算，就像下列循环一样：

for(int i = 0; i < 50; ++i)

a[i] = 3\*b[i] + 4\*c[i] + 5\*d[i];

因此，在eigen 中，你不必担心使用相当大的算术运算表达式，它会提供给eigen更多优化代码的机会。

5.转置和共轭

矩阵类的成员函数transpose(),conjugate(),adjoint(),分别对应矩阵的转置,共轭，共轭转置矩阵，特此说明adjoint()并不表示伴随矩阵，而是共轭转置矩阵！！！！

例子：

MatrixXcf a = MatrixXcf::Random(2,2);//生成随机的复数类型矩阵

cout << "Here is the matrix a\n" << a << endl;

cout << "Here is the matrix a^T\n" << a.transpose() << endl;

cout << "Here is the conjugate of a\n" << a.conjugate() << endl;

cout << "Here is the matrix a^\*\n" << a.adjoint() << endl;

//output

Here is the matrix a

(-0.211,0.68) (-0.605,0.823)

(0.597,0.566) (0.536,-0.33)

Here is the matrix a^T

(-0.211,0.68) (0.597,0.566)

(-0.605,0.823) (0.536,-0.33)

Here is the conjugate of a

(-0.211,-0.68) (-0.605,-0.823)

(0.597,-0.566) (0.536,0.33)

Here is the matrix a^\*

(-0.211,-0.68) (0.597,-0.566)

(-0.605,-0.823) (0.536,0.33)

对于实矩阵，是没有共轭矩阵的，同时它的共轭转置矩阵（adjoint()）等于它的转置(transpose()).

对于基本的算术运算，转置和共轭转置函数返回的是矩阵的引用，而不会实际转换矩阵对象。如果你对b做b=a.transpose(),这个将在求转置矩阵的同时，将结果赋值给b。但是如果将a=a.transpose(),eigen将会在计算a的转置完成之前开始赋值结果给a,因此，这样的赋值将不会将a替换成它的转置，而是：

Matrix2i a; a << 1, 2, 3, 4;

cout << "Here is the matrix a:\n" << a << endl;

a = a.transpose(); // !!! do NOT do this !!!

cout << "and the result of the aliasing effect:\n" << a << endl;

//output

Here is the matrix a:

1 2

3 4

and the result of the aliasing effect:

1 2

2 4

结果不再是a的转置，而是发生了混叠（aliasing issue）.在调试模式中，在到达断点之前，这样的错误很容易被检测到。

为了将a替换为a的转置矩阵，可以使用transposeInPlace()函数：

MatrixXf a(2,3); a << 1, 2, 3, 4, 5, 6;

cout << "Here is the initial matrix a:\n" << a << endl;

a.transposeInPlace();

cout << "and after being transposed:\n" << a << endl;

//output

Here is the initial matrix a:

1 2 3

4 5 6

and after being transposed:

1 4

2 5

3 6

同样地，对于共轭转置矩阵（adjoint()）也有类似的成员函数(adjointInPlace()).

6.矩阵-矩阵乘法和矩阵-向量乘法

矩阵乘法使用\*运算符；

双目运算符：a\*b

复合运算符：a\*=b

#include <iostream>

#include <Eigen/Dense>

using namespace Eigen;

int main()

{

Matrix2d mat;

mat << 1, 2,

3, 4;

Vector2d u(-1,1), v(2,0);

std::cout << "Here is mat\*mat:\n" << mat\*mat << std::endl;

std::cout << "Here is mat\*u:\n" << mat\*u << std::endl;

std::cout << "Here is u^T\*mat:\n" << u.transpose()\*mat << std::endl;

std::cout << "Here is u^T\*v:\n" << u.transpose()\*v << std::endl;

std::cout << "Here is u\*v^T:\n" << u\*v.transpose() << std::endl;

std::cout << "Let's multiply mat by itself" << std::endl;

mat = mat\*mat;

std::cout << "Now mat is mat:\n" << mat << std::endl;

}

//output

Here is mat\*mat:

7 10

15 22

Here is mat\*u:

1

1

Here is u^T\*mat:

2 2

Here is u^T\*v:

-2

Here is u\*v^T:

-2 -0

2 0

Let's multiply mat by itself

Now mat is mat:

7 10

15 22

说明，前述表达式m=m\*m可能会引起混叠的问题，但是对于矩阵乘法而言，不必担心：eigen将矩阵的乘法看作一种特殊的情况，它引入一个临时变量，因此它将编译成以下代码：

tmp = m\*m;

m = tmp;

如果你想让矩阵乘法安全的进行计算而没有混叠问题，你可以使用noalias()成员函数来避免临时变量的问题，例如：

c.noalias() += a \* b;

7.点乘和叉乘

点乘dot(),叉乘cross().点乘也可以使用u.adjoint()\*v。

例子：

#include <iostream>

#include <Eigen/Dense>

using namespace Eigen;

using namespace std;

int main()

{

Vector3d v(1,2,3);

Vector3d w(0,1,2);

cout << "Dot product: " << v.dot(w) << endl;//点乘

double dp = v.adjoint()\*w; // automatic conversion of the inner product to a scalar

cout << "Dot product via a matrix product: " << dp << endl;

cout << "Cross product:\n" << v.cross(w) << endl;//叉乘

}

//output

Dot product: 8

Dot product via a matrix product: 8

Cross product:

1

-2

1

注意：叉乘只能用于维数为3的向量，点乘使用于任何维数的向量。当使用复数时，第一个变量是共轭线性运算，第二个是线性运算。

8.基本的算术化简计算

eigen提供一些简化计算将给定的矩阵或向量编程单个值，比如对矩阵的所有元素求和sum(),求积prod(),求最大值maxCoeff()和求最小值minCoeff()：

#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;//对矩阵所有元素求和

cout << "Here is mat.prod(): " << mat.prod() << endl;//对矩阵所有元素求积

cout << "Here is mat.mean(): " << mat.mean() << endl;//对矩阵所有元素求平均值

cout << "Here is mat.minCoeff(): " << mat.minCoeff() << endl;//取矩阵的元素最小值

cout << "Here is mat.maxCoeff(): " << mat.maxCoeff() << endl;//取矩阵元素的最大值

cout << "Here is mat.trace(): " << mat.trace() << endl;//取矩阵元素的迹

}

//output

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

矩阵的迹返回的是矩阵对角线元素的和,等价于a.diagonal().sum().

同时求最大值和最小值的函数可以接受引用的实参，来表示其最大最小值的行数和列数：

Matrix3f m = Matrix3f::Random();

std::ptrdiff\_t i, j;//i,j是一个整型类型

float minOfM = m.minCoeff(&i,&j);//矩阵可以接受两个引用参数

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

cout << "Its minimum coefficient (" << minOfM

<< ") is at position (" << i << "," << j << ")\n\n";//输出最小值所在行数列数

RowVector4i v = RowVector4i::Random();

int maxOfV = v.maxCoeff(&i);//向量只接受一个引用参数

cout << "Here is the vector v: " << v << endl;

cout << "Its maximum coefficient (" << maxOfV

<< ") is at position " << i << endl;//输出最大值所在列数

//output

Here is the matrix m:

0.68 0.597 -0.33

-0.211 0.823 0.536

0.566 -0.605 -0.444

Its minimum coefficient (-0.605) is at position (2,1)

Here is the vector v: 1 0 3 -3

Its maximum coefficient (3) is at position 2

9.运算的有效性

eigen库会检查你定义的运算。通常它在编译时检查并产生错误信息。这些错误信息可能很长很丑，但是eigen将重要信息用大写字母来显示出，例如：

Matrix3f m;

Vector4f v;

v = m\*v; // Compile-time error: YOU\_MIXED\_MATRICES\_OF\_DIFFERENT\_SIZES

在许多情况下，当使用动态绑定矩阵时，编译器将不会在编译时检查，eigen将会在运行时检查，yejiuis说程序有可能因为不合法的运算而中断。

MatrixXf m(3,3);

VectorXf v(4);

v = m \* v; // Run-time assertion failure here: "invalid matrix product"