shuzfan的专栏

目录视图

个人资料

shuzfan

关注 发私信

访问： 131616次
积分： 2279
等级： BLOG 5
排名： 第14344名

原创： 67篇 转载： 7篇
译文： 1篇 评论： 265条

文章搜索

文章分类

计算机视觉 (2)
MATLAB (3)
程序、数据资料 (12)
深度学习基础 (10)
caffe (11)
人脸识别 (3)
神经网络压缩与加速 (7)
人脸检测(目标检测) (15)
人脸关键点检测 (1)
特征选择 (1)
C++ (7)
排序算法 (7)

文章存档

2017年03月 (2)
2017年02月 (8)
2017年01月 (5)
2016年12月 (2)
2016年11月 (7)

展开

Eigen矩阵运算库使用记录

2016-08-30 10:42 1876人阅读 评论

分类： 程序、数据资料（11）

最近一直在做工程上的事情，比较多的使用了Eigen矩阵运算库。

简单说一下Eigen的特点：

(1) 使用方便、无需预编译，调用开销小

(2) 函数丰富，风格有点近似MATLAB，易上手；

(3) 速度中规中矩，比**OpenCV**快，比MKL、openBLAS慢；

Eigen3.3版本链接 http://eigen.tuxfamily.org/index.php?title=Main_Page

注：绝大部分使用说明和示例都可以在官网上找到，所以有时候不需要纠结百度到的与实际不符，可以直接官网or谷歌

使用方法很简单：下载Eigen后解压，然后包含解压路径，最后只需要在程序里引用头文件

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

基本使用方法如下：

原址链接为http://eigen.tuxfamily.org/dox/AsciiQuickReference.txt

矩阵定义

```
#include <Eigen/Dense>

Matrix<double, 3, 3> A;                // Fixed rows and cols. Same as Matrix3d.
Matrix<double, 3, Dynamic> B;          // Fixed rows, dynamic cols.
Matrix<double, Dynamic, Dynamic> C;    // Full dynamic. Same as MatrixXd.
Matrix<double, 3, 3, RowMajor> E;      // Row major; default is column-major.
Matrix3f P, Q, R;                      // 3x3 float matrix.
Vector3f x, y, z;                      // 3x1 float matrix.
RowVector3f a, b, c;                   // 1x3 float matrix.
VectorXd v;                            // Dynamic column vector of doubles
```

基本使用方法

```
// Basic usage
// Eigen          // Matlab           // comments
```

```
x.size()          // length(x)        // vector size
C.rows()          // size(C,1)        // number of rows
C.cols()          // size(C,2)        // number of columns
x(i)              // x(i+1)           // Matlab is 1-based
C(i,j)            // C(i+1,j+1)       //

A.resize(4, 4);   // Runtime error if assertions are on.
B.resize(4, 9);   // Runtime error if assertions are on.
A.resize(3, 3);   // Ok; size didn't change.
B.resize(3, 9);   // Ok; only dynamic cols changed.

A << 1, 2, 3,     // Initialize A. The elements can also be
     4, 5, 6,     // matrices, which are stacked along cols
     7, 8, 9;     // and then the rows are stacked.
B << A, A, A;     // B is three horizontally stacked A's.
A.fill(10);       // Fill A with all 10's.
```

特殊矩阵生成

```
// Eigen                            // Matlab
MatrixXd::Identity(rows,cols)       // eye(rows,cols)
C.setIdentity(rows,cols)            // C = eye(rows,cols)
MatrixXd::Zero(rows,cols)           // zeros(rows,cols)
C.setZero(rows,cols)                // C = ones(rows,cols)
MatrixXd::Ones(rows,cols)           // ones(rows,cols)
C.setOnes(rows,cols)                // C = ones(rows,cols)
MatrixXd::Random(rows,cols)         // rand(rows,cols)*2-1        // MatrixXd::Random
C.setRandom(rows,cols)              // C = rand(rows,cols)*2-1
VectorXd::LinSpaced(size,low,high)   // linspace(low,high,size)'
v.setLinSpaced(size,low,high)        // v = linspace(low,high,size)'
```

矩阵块操作

```
// Matrix slicing and blocks. All expressions listed here are read/write.
// Templated size versions are faster. Note that Matlab is 1-based (a size N
// vector is x(1)...x(N)).
// Eigen                           // Matlab
x.head(n)                          // x(1:n)
x.head<n>()                        // x(1:n)
x.tail(n)                          // x(end - n + 1: end)
x.tail<n>()                        // x(end - n + 1: end)
x.segment(i, n)                    // x(i+1 : i+n)
x.segment<n>(i)                    // x(i+1 : i+n)
P.block(i, j, rows, cols)          // P(i+1 : i+rows, j+1 : j+cols)
P.block<rows, cols>(i, j)          // P(i+1 : i+rows, j+1 : j+cols)
P.row(i)                           // P(i+1, :)
P.col(j)                           // P(:, j+1)
P.leftCols<cols>()                 // P(:, 1:cols)
P.leftCols(cols)                   // P(:, 1:cols)
P.middleCols<cols>(j)              // P(:, j+1:j+cols)
P.middleCols(j, cols)              // P(:, j+1:j+cols)
P.rightCols<cols>()                // P(:, end-cols+1:end)
P.rightCols(cols)                  // P(:, end-cols+1:end)
P.topRows<rows>()                  // P(1:rows, :)
P.topRows(rows)                    // P(1:rows, :)
P.middleRows<rows>(i)              // P(i+1:i+rows, :)
P.middleRows(i, rows)              // P(i+1:i+rows, :)
P.bottomRows<rows>()               // P(end-rows+1:end, :)
P.bottomRows(rows)                 // P(end-rows+1:end, :)
P.topLeftCorner(rows, cols)        // P(1:rows, 1:cols)
P.topRightCorner(rows, cols)       // P(1:rows, end-cols+1:end)
P.bottomLeftCorner(rows, cols)     // P(end-rows+1:end, 1:cols)
P.bottomRightCorner(rows, cols)    // P(end-rows+1:end, end-cols+1:end)
P.topLeftCorner<rows,cols>()       // P(1:rows, 1:cols)
P.topRightCorner<rows,cols>()      // P(1:rows, end-cols+1:end)
P.bottomLeftCorner<rows,cols>()    // P(end-rows+1:end, 1:cols)
P.bottomRightCorner<rows,cols>()   // P(end-rows+1:end, end-cols+1:end)
```

矩阵元素交换以及转置等

```
// Of particular note is Eigen's swap function which is highly optimized.
// Eigen                           // Matlab
R.row(i) = P.col(j);               // R(i, :) = P(:, i)
R.col(j1).swap(mat1.col(j2));      // R(:, [j1 j2]) = R(:, [j2, j1])

```

```
// Views, transpose, etc; all read-write except for .adjoint().
// Eigen                           // Matlab
R.adjoint()                        // R'
R.transpose()                      // R.' or conj(R')
R.diagonal()                       // diag(R)
x.asDiagonal()                     // diag(x)
R.transpose().colwise().reverse(); // rot90(R)
R.conjugate()                      // conj(R)
```

矩阵四则运算

```
// All the same as Matlab, but matlab doesn't have *= style operators.
// Matrix-vector.  Matrix-matrix.   Matrix-scalar.
y  = M*x;          R  = P*Q;        R  = P*s;
a  = b*M;          R  = P - Q;      R  = s*P;
a *= M;            R  = P + Q;      R  = P/s;
                   R *= Q;          R  = s*P;
                   R += Q;          R *= s;
                   R -= Q;          R /= s;
```

单个元素操作

```
// Vectorized operations on each element independently
// Eigen                  // Matlab
R = P.cwiseProduct(Q);    // R = P .* Q
R = P.array() * s.array();// R = P .* s
R = P.cwiseQuotient(Q);   // R = P ./ Q
R = P.array() / Q.array();// R = P ./ Q
R = P.array() + s.array();// R = P + s
R = P.array() - s.array();// R = P - s
R.array() += s;           // R = R + s
R.array() -= s;           // R = R - s
R.array() < Q.array();    // R < Q
R.array() <= Q.array();   // R <= Q
R.cwiseInverse();         // 1 ./ P
R.array().inverse();      // 1 ./ P
R.array().sin()           // sin(P)
R.array().cos()           // cos(P)
R.array().pow(s)          // P .^ s
R.array().square()        // P .^ 2
R.array().cube()          // P .^ 3
R.cwiseSqrt()             // sqrt(P)
R.array().sqrt()          // sqrt(P)
R.array().exp()           // exp(P)
R.array().log()           // log(P)
R.cwiseMax(P)             // max(R, P)
R.array().max(P.array())  // max(R, P)
R.cwiseMin(P)             // min(R, P)
R.array().min(P.array())  // min(R, P)
R.cwiseAbs()              // abs(P)
R.array().abs()           // abs(P)
R.cwiseAbs2()             // abs(P.^2)
R.array().abs2()          // abs(P.^2)
(R.array() < s).select(P,Q);  // (R < s ? P : Q)
```

矩阵缩减

```
// Reductions.
int r, c;
// Eigen                  // Matlab
R.minCoeff()              // min(R(:))
R.maxCoeff()              // max(R(:))
s = R.minCoeff(&r, &c)    // [s, i] = min(R(:)); [r, c] = ind2sub(size(R), i);
s = R.maxCoeff(&r, &c)    // [s, i] = max(R(:)); [r, c] = ind2sub(size(R), i);
R.sum()                   // sum(R(:))
R.colwise().sum()         // sum(R)
R.rowwise().sum()         // sum(R, 2) or sum(R')'
R.prod()                  // prod(R(:))
R.colwise().prod()        // prod(R)
R.rowwise().prod()        // prod(R, 2) or prod(R')'
R.trace()                 // trace(R)
R.all()                   // all(R(:))
R.colwise().all()         // all(R)
R.rowwise().all()         // all(R, 2)
R.any()                   // any(R(:))
```

```
R.colwise().any()          // any(R)
R.rowwise().any()          // any(R, 2)
```

矩阵点乘及归一化

```
// Dot products, norms, etc.
// Eigen                   // Matlab
x.norm()                   // norm(x).    Note that norm(R) doesn't work in Eigen.
x.squaredNorm()            // dot(x, x)   Note the equivalence is not true for complex
x.dot(y)                   // dot(x, y)
x.cross(y)                 // cross(x, y) Requires #include <Eigen/Geometry>
```

矩阵类型转换

```
//// Type conversion
// Eigen                           // Matlab
A.cast<double>();                  // double(A)
A.cast<float>();                   // single(A)
A.cast<int>();                     // int32(A)
A.real();                          // real(A)
A.imag();                          // imag(A)
// if the original type equals destination type, no work is done

// Note that for most operations Eigen requires all operands to have t          e:
MatrixXf F = MatrixXf::Zero(3,3);
A += F;                 // illegal in Eigen. In Matlab A = A+F is allow
A += F.cast<double>(); // F converted to double and then added (generally, conversion
```

内存映射创建矩阵

```
// Eigen can map existing memory into Eigen matrices.
float array[3];
Vector3f::Map(array).fill(10);            // create a temporary Map over array and set
int data[4] = {1, 2, 3, 4};
Matrix2i mat2x2(data);                    // copies data into mat2x2
Matrix2i::Map(data) = 2*mat2x2;           // overwrite elements of data with 2*mat2x2
MatrixXi::Map(data, 2, 2) += mat2x2;      // adds mat2x2 to elements of data (alternat
```

解方程

```
// Solve Ax = b. Result stored in x. Matlab: x = A \ b.
x = A.ldlt().solve(b));  // A sym. p.s.d.    #include <Eigen/Cholesky>
x = A.llt() .solve(b));  // A sym. p.d.      #include <Eigen/Cholesky>
x = A.lu()  .solve(b));  // Stable and fast. #include <Eigen/LU>
x = A.qr()  .solve(b));  // No pivoting.     #include <Eigen/QR>
x = A.svd() .solve(b));  // Stable, slowest. #include <Eigen/SVD>
// .ldlt() -> .matrixL() and .matrixD()
// .llt()  -> .matrixL()
// .lu()   -> .matrixL() and .matrixU()
// .qr()   -> .matrixQ() and .matrixR()
// .svd()  -> .matrixU(), .singularValues(), and .matrixV()
```

特征值

```
// Eigenvalue problems
// Eigen                          // Matlab
A.eigenvalues();                  // eig(A);
EigenSolver<Matrix3d> eig(A);     // [vec val] = eig(A)
eig.eigenvalues();                // diag(val)
eig.eigenvectors();               // vec
// For self-adjoint matrices use SelfAdjointEigenSolver<>
```

下面是我实际遇到的补充一下：

求广义逆矩阵

```
//Eigen中并没有求广义逆的函数，这里用SVD实现，数据类型大家可以看需修改为doubel
using Eigen::Dynamic;
using Eigen::Matrix;
using Eigen::RowMajor;
typedef Matrix<float, Dynamic, Dynamic, RowMajor> MatXf;

```

```
MatXf pinv(MatXf x)
{
    JacobiSVD<MatXf> svd(x,ComputeFullU | ComputeFullV);
    float  pinvtoler=1.e-8; //tolerance
    MatXf singularValues_inv = svd.singularValues();
    for ( long i=0; i<x.cols(); ++i) {
        if ( singularValues_inv(i) > pinvtoler )
            singularValues_inv(i)=1.0/singularValues_inv(i);
        else singularValues_inv(i)=0;
    }
    return svd.matrixV()*singularValues_inv.asDiagonal()*svd.matrixU().transpose();
}
```

常用的和MATLAB类似的函数实现如下：

原址链接http://igl.ethz.ch/projects/libigl/matlab-to-eigen.html

MATLAB	Eigen
`[Y,IX] = sort(Y,dim,mode)`	`igl::sort(X,dim,mode,Y,IX)`
`B(i:(i+w),j:(j+h)) = A(x:(x+w),y:(y+h))`	`B.block(i,j,w,h) = A.block(i,j,w,h)`
`max(A(:))`	`A.maxCoeff()`
`min(A(:))`	`A.minCoeff()`
`eye(w,h)`	`MatrixXd::Identity(w,h), MatrixXf::Identity(w,h), etc.`
`A(i:(i+w),j:(j+h)) = eye(w,h)`	`A.setIdentity()`
`[I,J,V] = find(X)`	`igl::find(X,I,J,V)`
`X(:,j) = X(:,j) + x`	`X.col(j).array() += x`

`Acol_sum = sum(A,1)`
`Arow_sum = sum(A,2)`
`Adim_sum = sum(Asparse,dim)`
	`Acol_sum = A.colwise().sum()`
	`Arow_sum = A.rowwise().sum()`
	`igl::sum(Asparse,dim,Adim_sum)`

`D = diag(M)`	`igl::diag(M,D)`
`M = diag(D)`	`igl::diag(D,M)`
`[Y,I] = max(X,[],dim)`	`igl::mat_max(X,dim,Y,I)`

`Y = max(X,[],1)`
`Y = max(X,[],2)`
`Y = min(X,[],1)`
`Y = min(X,[],2)`
	`Y = X.colwise().maxCoeff()`
	`Y = X.rowwise().maxCoeff()`
	`Y = X.colwise().minCoeff()`
	`Y = X.rowwise().minCoeff()`

`C = A.*B`	`C = (A.array() * B.array()).matrix()`
`C = A.^b`	`C = A.array().pow(b).matrix()`
`A(B == 0) = C(B==0)`	`A = (B.array() == 0).select(C,A)`
`C = A + B'`	`SparseMatrixType BT = B.transpose()` `SparseMatrixType C = A+BT;`
`[L,p] = chol(A)`	`SparseLLT<SparseMatrixType> A_LLT(A.template triangularView<Lower>())` `SparseMatrixType L = A_LLT.matrixL();bool p = (L*0).eval().nonZeros()==0;`

`X = U\(L\b)`	`X = b;`
	`L.template triangularView<Lower>().solveInPlace(X);`
	`U.template triangularView<Upper>().solveInPlace(X);`

`B = repmat(A,i,j)`	`igl::repmat(A,i,j,B)`
`I = low:step:hi`	`igl::colon(low,step,hi,I)`
`O = ones(m,n)`	`Matrix* O = Matrix*::Ones(m,n)`
`O = zeros(m,n)`	`Matrix* O = Matrix*::Zero(m,n)`
`B = A(I,J)` `B = A(I,:)`	`igl::slice(A,I,J,B)` `B = igl::slice(A,I,igl::colon(0,A.cols()-1))`
`B(I,J) = A` `B(I,:) = A`	`igl::slice_into(A,I,J,B)` `B = igl::slice_into(A,I,igl::colon(0,B.cols()-1))`
`M = mode(X,dim)`	`igl::mode(X,dim,M)`
`B = arrayfun(FUN, A)`	`B = A.unaryExpr(ptr_fun(FUN))`

`B = fliplr(A)`
`B = flipud(A)`
	`B = A.rowwise().reverse().eval()`
	`B =`
	`A.colwise().reverse().eval()`

`B = IM(A)`	`B = A.unaryExpr(bindlst(mem_fun(`
	`static_cast<VectorXi::Scalar&(VectorXi::*)(VectorXi::Index)>`
	`(&VectorXi::operator())), &IM)).eval();`

`A = sparse(I,J,V)`	`// build std::vector<Eigen::Triplet> IJV` `A.setFromTriplets(IJV);`
`A = min(A,c);`	`C.array() = A.array().min(c);`

`IP = I(P==0);`	`VectorXi IP = I;`
	`IP.conservativeResize(stable_partition(`
	`IP.data(),`
	`IP.data()+IP.size(),`
	`[&P](int i){return P(i)==0;})-IP.data());`

`a = any(A(:))`	`bool a = any_of(A.data(),A.data()+A.size(),[](bool a){ return a;});`