**Matrix and Vector Algebra**

**(matrix.c)**

**Technical Note**

Document XXX

Version: Draft

Authors: Mark Pedley and Michael Stanley

Date: xx Aug 2014



Table of Contents

[1 Introduction 4](#_Toc396290874)

[1.1 Summary 4](#_Toc396290875)

[1.2 Functions 4](#_Toc396290876)

[2 Primer on Matrix and Vector Algebra 5](#_Toc396290877)

[2.1 Matrix Identities 5](#_Toc396290878)

[2.2 Rotation Matrices 5](#_Toc396290879)

[2.3 Eigenvectors and Eigenvalues 6](#_Toc396290880)

[2.4 Vector Identities 7](#_Toc396290881)

[3 EigenAnalysis by Jacobi Algorithm 9](#_Toc396290882)

[3.1 Introduction 9](#_Toc396290883)

[3.2 Givens Rotation Matrix 9](#_Toc396290884)

[3.3 Determining the Givens Rotation Angle 10](#_Toc396290885)

[3.4 The Jacobi Algorithm 11](#_Toc396290886)

[4 Normalization of a Rotation Matrix 14](#_Toc396290887)

**Glossary**

General matrices

Transpose of matrix

Inverse of matrix

Determinant of matrix

Identity matrix

Rotation matrix

Rotation matrix about axis by angle

*i-th* eigenvector

Rotation angles

Small rotation angles

*i-th* eigenvalue

Diagonal eigenvalue matrix

Matrix of column eigenvectors

Jacobi rotation angle

## Introduction

### Summary

This Application Note provides an introduction to matrix algebra in section 2. With this introduction, the matrix functions in matrix.c should be self-explanatory with the exception of:

a) the eigensolver function eigencompute which uses the Jacobi rotation algorithm documented in section 3

b) the rotation matrix re-normalization function fmatrixAeqRenormRotA which is documented in section 4.

### Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| void f3x3matrixAeqI(float A[][3]); | Set the matrix A(3 by 3) to the identity matrix I(3) |
| void fmatrixAeqI(float \*A[], int16 rc); | Set the matrix A(rc by rc) to the identity matrix I(rc) |
| void f3x3matrixAeqScalar(float A[][3], float Scalar); | Sets the matrix A(3 by 3) to the constant value Scalar.  Always called in practice with Scalar=0.0F to zero the matrix. |
| void f3x3matrixAeqInvSymB(float A[][3], float B[][3]); | Sets A(3 by 3) to the inverse of the symmetric matrix B(3x3) |
| void f3x3matrixAeqAxScalar(float A[][3], float Scalar); | A(3 by 3) = A(3 by 3) \* Scalar  Used for normalization purposes. |
| void f3x3matrixAeqMinusA(float A[][3]); | Negates all elements of the matrix A(3 by 3) |
| float f3x3matrixDetA(float A[][3]); | Returns the determinant of the matrix A(3 by 3) |
| void eigencompute(float A[][10], float eigval[], float eigvec[][10], int8 n); | Computes the eigenvalues and eigenvectors of the square matrix m(n by n) stored in the upper left of a 10x10 array. |
| void fmatrixAeqInvA(float \*A[], int8 iColInd[], int8 iRowInd[], int8 iPivot[], int8 isize); | Computes the in-site inverse of the matrix A(isize by isize) |
| void fmatrixAeqRenormRotA(float A[][3]); | Function removes rounding errors from the rotation matrix A(3 by 3) and re-constructs an exact orthonormal rotation matrix. |

## Primer on Matrix and Vector Algebra

### Matrix Identities

For matrices and of arbitrary dimension:

Eq 2.1.1

Eq 2.1.2

Eq 2.1.3

If the product is a scalar:

Eq 2.1.4

If and are square and invertible, the inverse of their product satisfies:

Eq 2.1.5

The inverse and transpose operations commute for an invertible square matrix:

Eq 2.1.6

An orthogonal matrix is a real square matrix whose columns and columns are orthonormal. A consequence is that the transpose of an orthonormal matrix is its inverse.

Eq 2.1.7

### Rotation Matrices

Rotation matrices are square matrices with determinant of +1 and with orthonormal columns and rows.

Eq 2.2.1

Eq 2.2.2

Rotation matrices about a given axis with specified angle can be defined as either the rotation matrix rotating the coordinate frame or rotating a vector in the coordinate frame. The two are related by having the same axis but opposite rotation angles. The convention used in Freescale's Sensor fusion Toolbox is that the coordinate system is rotated about the rotation vector.

Rotation matrices do not commute and the order of rotations must always be specified.

Eq 2.2.3

Euler's rotation theorem states that any sequence of rotation matrices can be represented by a single rotation matrix with axis and angle to be determined. The general matrix for a rotation of a coordinate frame about (normalized) axis by angle is the Rodrigues matrix:

Eq 2.2.4

Inspection of equation 2.2.4 shows that the rotation matrix is symmetric when which occurs at and (180o) rotations.

Eq 2.2.5

Eq 2.2.6

Evaluating for aligned along the *x*, *y* and *z* axes produces the Cartesian rotation matrices:

Eq 2.2.7

Eq 2.2.8

Eq 2.2.9

For a small rotation angles , the general rotation matrix simplifies to:

Eq 2.2.10

Rotation matrices do commute in the limit of small angles and :

Eq 2.2.11

Eq 2.2.12

### Eigenvectors and Eigenvalues

The *i-th* column eigenvector and eigenvalue of any square matrix are defined as satisfying:

Eq 2.3.1

It is commonplace to create a eigenvector matrix from the individual column eigenvectors :

Eq 2.3.2

Equation 2.3.1 can then be written in the form:

Eq 2.3.3

where is the diagonal matrix of the eigenvalues:

Eq 2.3.4

The rotation axis of a rotation matrix is, by definition, invariant under rotation and is therefore the normalized eigenvector corresponding to eigenvalue 1:

Eq 2.3.5

This can be confirmed for the general rotation matrix of equation 2.2.4:

Eq 2.3.6

Eq 2.3.7

### Vector Identities

Repeated application of Pythagoras' theorem relates the transpose product of any vector with itself to its magnitude squared.

Eq 2.4.1

The transpose product of two vectors is often written as the scalar or dot product. The cosine rule of trigonometry relates the scalar product to the cosine of the angle between the two vectors as:

Eq 2.4.2

Eq 2.4.3

The vector or cross product between two vectors and is defined to be a new vector with magnitude in a direction orthogonal to both and :

Eq 2.4.4

In Cartesian coordinates, the vector product can be written as:

Eq 2.4.5

An alternative formulation of the vector product is matrix multiplication by the 3x3 matrix defined as:

Eq 2.4.6

A useful result for the vector product of three vectors is:

Eq 2.4.7

## EigenAnalysis by Jacobi Algorithm

### Introduction

This section documents the Jacobi algorithm underlying the function eigencompute which computes the eigenvalues and eigenvectors of a symmetric matrix by successive diagonalization by Givens rotation matrices.

The matrix of real eigenvectors anddiagonal matrix of real eigenvaluesof the real symmetric matrix satisfy:

Eq 3.1.1

Pre-and post multiplying equation 3.1.1 by a sequence of plane rotation matrices and(termed Givens rotation matrices) selected to zero the off-diagonal elements and diagonalize gives:

Eq 3.1.2

Eq 3.1.3

The eigenvalues of the matrix are then the elements of the diagonal matrix derived by zeroing off-diagonal elements in and the matrix of eigenvectors is the product of the sequence of matrices used to perform the diagonalization. This diagonalization strategy is used by the Jacobi rotation algorithm successively applying Givens rotation matrices.

### Givens Rotation Matrix

The Givens matrix for rotation angle has form:

Eq 3.2.1

All the diagonal elements are zero except for the two elements at positions and . All off-diagonal elements are zero except for the two elements at positions and . The Givens rotation matrix is orthonormal as required for a rotation matrix.

A general matrix is transformed by pre- and post-multiplication by the Givens rotation matrix with non-zero elements in rows and as:

Eq 3.2.2

The elements in changed by this operation are:

Eq 3.2.3

The changed elements of equation 3.2.3 are:

Eq 3.2.4

Eq 3.2.5

Eq 3.2.6

Eq 3.2.7

Eq 3.2.8

### Determining the Givens Rotation Angle

The required rotation angle is that which zeroes out element :

Eq 3.3.1

Standard trigonometry identities allow the cotangent of twice the rotation angle to be written as:

Eq 3.3.2

Combining equations 3.3.1 and 3.3.2 gives the rotation angle as:

Eq 3.3.3

Eq 3.3.4

Eq 3.3.5

Taking the positive square root for gives:

Eq 3.3.6

Eq 3.3.7

Taking the negative square root gives:

Eq 3.3.8

Eq 3.3.9

For negative, the smaller magnitude of the two solutions is:

Eq 3.3.10

For positive, the smaller magnitude of the two solutions is:

Eq 3.3.11

In both cases:

Eq 3.3.12

If is so large that squared would overflow, the alternative is:

Eq 3.3.13

A trigonometric identity used later is:

Eq 3.3.14

### The Jacobi Algorithm

To avoid roundoff error, iterative updates are used for equations 3.2.4 to 3.2.8:

**Equation 3.2.8:**

By definition, the rotation angle is selected so that equation 3.2.8 gives zero .

Eq 3.4.1

Eq 3.4.2

Eq 3.4.3

**Equation 3.2.6:**

Eq 3.4.4

Substituting equation 3.4.3 gives:

Eq 3.4.5

Eq 3.4.6

Eq 3.4.7

**Equation 3.2.7:**

By definition of the rotation angle which zeroes equation 3.2.8:

Eq 3.4.8

Substituting into equation 3.2.7 gives: Eq 3.4.9

Eq 3.4.10

Eq 3.4.11

**Equation 3.2.4:**

Eq 3.4.12

Eq 3.4.13

Substituting equation 3.3.14 gives:

Eq 3.4.14

**Equation 3.2.5:**

Eq 3.4.15

Eq 3.4.16

Substituting equation 3.3.14 gives:

Eq 3.4.17

## Normalization of a Rotation Matrix

The rows and columns of a rotation matrix are orthonormal. After several thousand iterative rotations, a rotation matrix can become degraded as a result of numerical rounding errors. The simple method below permits the re-normalization of a rotation matrix and is used in function fmatrixAeqRenormRotA.

Given an initial rotation matrix with column vectors , , :

Eq 4.1.1

Step 1: The column vector is the vector normalized:

Eq 4.1.2

Step 2: The column vector is set to:

Eq 4.1.3

The column vector is obviously normalized and is easily proved to be orthogonal to :

Eq 4.1.4

Step 3: The column vector is set to the unit vector orthogonal to and :

Eq 4.1.5

There is no need to explicitly normalize the vector since and are unit vectors and is orthogonal to .

The re-normalized rotation matrix is:

Eq 4.1.6