CEE 532 – Developing Software Engineering Applications

Project 1 – A Matrix Toolbox

Michael Justice

10/10/15

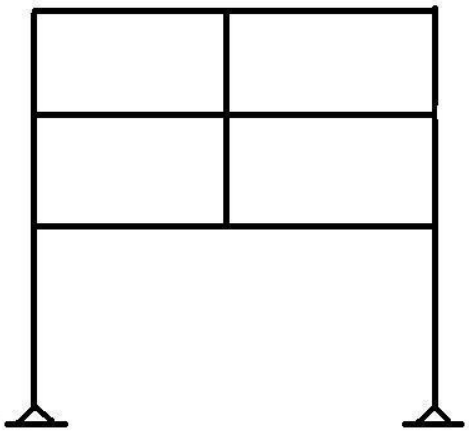

Table of Contents

Matrices and Vectors (A Definition)

In order to effectively perform operations involving vectors and matrices, we must first define a matrix. A matrix is a two-dimensional rectangular mathematical structure that contains numerical values located by a row and column index. The use of a matrix is to store values in a specific order so that the values can be made of use to serve a particular purpose. In the case of this report (and future others), the main purpose of constructing and operating within the confines of matrix algebra leads to efficiency in solving many tedious, linear algebraic equations. Matrices can be divided into subsets called vectors, specifically row and column vectors. Vectors can define the direction and magnitude of 3 dimensional quantities, and specifically in the context of engineering, is useful in determining the position of one point in space relative to another. Note that vectors are not limited to storing only three values, but when a vector contains more than 3, it is customary to label such a vector as a 1-dimensional matrix.

An example of several vectors is shown below, as well some special matrices which serve a great purpose in solving linear algebraic equations.

The Row Vector

$$\mathbf{a_{1xn}} = \{a_1, a_2, a_3, \cdots a_n\}$$

A row vector is a vector that contains values (or elements) along a horizontal.

The Column Vector

$$\mathbf{a_{mx1}} = \begin{Bmatrix} a_1 \\ \vdots \\ a_m \end{Bmatrix}$$

A column vector is a vector that contains values (or elements) along a vertical.

The Null Vector

The vector (row or column) that contains values equal to zero is called a null vector.

$$\mathbf{a_{1xn}} = \{0, 0, 0, \cdots 0\}$$

Square Matrix

The square matrix is a matrix that has a number of rows equal to the number of columns.

$$\mathbf{A_{3x3}} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 5 & 6 & 7 \end{bmatrix}$$

Symmetric Matrix

A symmetric matrix is one in which $A_{ij} = A_{ji}$. This is best described by illustration.

$$\mathbf{A_{3x3}} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}$$

Diagonal Matrix

A matrix that contains values only along a diagonal (and all other entries are zero) is known as a diagonal matrix. An example is as shown.

$$\mathbf{A_{3x3}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

Identity Matrix

The identity matrix is a special case of the diagonal matrix in which the non-zero entries are all one.

$$\mathbf{A_{3x3}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Upper Triangular Matrix

The upper triangular matrix contains zero entries below the main diagonal, and has at least one non-zero entry above the main diagonal.

$$\mathbf{A_{3x3}} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 5 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

Lower Triangular Matrix

Conversely, the lower triangular matrix contains zero entries above the main diagonal, and has at least one non-zero entry below the main diagonal.

$$\mathbf{A_{3x3}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

Positive Definite Matrix

A positive-definite matrix is one in which, if square, contains eigenvalues that are greater than zero (positive).

Hermitian Matrix

A Hermitian matrix is one in which, if the matrix itself is of real values and square, is symmetric. If the matrix is not of real values but still square, a Hermitian matrix is one that is equal to its complex-conjugate of its transpose.

With the definitions of vectors and matrices now known, we can move on to operations involving matrices and vectors.

The Operations (And Algorithms) of the Matrix Toolbox

Operations between matrices, between vectors, or between *both* matrices and vectors are essential to matrix algebra, and all rules and constraints must be understood for implementation to be successful. Each matrix and vector operation that is specific to the contents of this report will be discussed, along with the necessary algorithm to implement the corresponding operation.

Vector Operations

Add and Subtract (Vector)

Column and row vector addition and subtraction are shown below. Since a vector is one-dimensional, the row or the column index (i or j) is only of note for either a column or row vector, respectively.

$$\mathbf{c_{mx1}} = \mathbf{a_{mx1}} \pm \mathbf{b_{mx1}}$$

$$\mathbf{c_{1xn}} = \mathbf{a_{1xn}} \pm \mathbf{b_{1xn}}$$

Algorithm:

$$c_i = a_i \pm b_i$$

$$c_j = a_j \pm b_j$$

Dot Product

The dot product (or inner product) of two vectors can be used to determine the angle between the two vectors. This has quite a significant in structural engineering software, as the dot product can be used to explain the geometric relationship between member elements (vectors). The result of a dot product between two vectors is a scalar. The dot product can also be defined as the length of the projection of vector $\mathbf{a}$ onto $\mathbf{b}$ multiplied by the length (or magnitude) of $\mathbf{b}$. The constraint for this operation to take place is that both vectors must be of the same size. *Only the column vector* will be shown here out to explain the theory of vector operations.

$$c = \mathbf{a_{mx1}} \cdot \mathbf{b_{mx1}}$$

Algorithm:

$$c = \sum_{k=1}^{m} a_i * b_i$$

As shown above, the dot product is simply the sum of the products of the row indices of each vector.

Normalize

The normalization of a vector is simply a vector that has its elements divided by the magnitude of the vector. In other words, this is the unit vector operation. The constraint for such an operation is that if the vector a is a null vector, division of zero will occur, which will cause an error. Therefore, the normalized vector will not be computed properly.

$$normalized\ \mathbf{a}\ (or\ \boldsymbol{a}^*) = \frac{\mathbf{a}}{\|\mathbf{a}\|}$$

Algorithm:

$$\text{If } \mathbf{a}\ is\ null, \text{stop.}$$

$$\|\mathbf{a}\| = \sqrt{\sum_{k=1}^{m} a_i * a_i}$$

$$\mathbf{a}^* = \frac{\mathbf{a}}{\|\mathbf{a}\|}$$

Scale

The scaling of a vector is defined by multiplying a vector by a scaling constant c. The result is a vector that is scaled by the constant c.

$$\mathbf{a} = \mathrm{c} * \boldsymbol{a}$$

Algorithm:

$$a_i = c * a_i$$

MaxValue

The maximum value (or element) of a vector is the maximum value with respect to the sign of the entries of the vector. In other words,

$$max\ value = \max(\mathbf{a})$$

Algorithm:

$$max = a_1$$

$$If\ a_i \geq \max, then \max = a_i$$

The algorithm is simple: start with an initial max that is of the first entry of a, then check all entries of a to be larger than the initial max. If an entry is larger than the initial max, then the "initial" max is set equal to that entry.

MinValue

The minimum value of a vector can be determined in a similar fashion to that of the MaxValue operation. Like the MaxValue operation, the minimum value is determined with respect to the sign of the entries of the vector.

$$min\ value = \min(\mathbf{a})$$

Algorithm:

$$min = a_1$$

$$If\ a_i \leq \min, then \min = a_i$$

TwoNorm

The two-norm of a vector is simply the magnitude of the vector. The magnitude has already been defined inexplicitly in the explanation of the Normalize operation above.

$$magnitude = \|\mathbf{a}\|$$

Algorithm:

$$\|\mathbf{a}\| = \sqrt{\sum_{k=1}^{m} a_i * a_i}$$

The magnitude of a vector can be defined as the 'largeness' of a vector. This is primarily useful in quantifying the accuracy of the solution to algebraic equations given a residual vector.

MaxNorm

The maximum norm of a vector is simply the largest entry of a vector in terms of the magnitude of the entry (ill-respect to sign). Therefore, this is not the same as the MaxValue operation.

$$max\ norm = \|\mathbf{a}\|_\infty$$

Algorithm:

$$max\ norm = abs(a_1)$$

$$iteration\ max\ norm = abs(a_i)$$

$$If\ \text{iteration max norm} \geq \text{max norm}, then\ \text{max norm} = \text{iteration max norm}$$

Similar to the MaxValue operation, the iteration max norm is checked against the "initial" max norm. Note that all entries within vector **a** are first made absolute and then checked if one is greater than the other.

CrossProduct

The cross product of two vectors in three dimensions is as shown below. Note that the result is a third vector **c**.The constraint for this operation to take place is that the two vectors must be three-dimensional.

$$\mathbf{c}_{mx1} = \mathbf{a}_{mx1} \times \mathbf{b}_{mx1}$$

Algorithm*:*

$$c_1 = a_2 b_3 - a_3 b_2$$

$$c_2 = a_3 b_1 - a_1 b_3$$

$$c_3 = a_1 b_2 - a_2 b_1$$

The magnitude of the cross product vector can be used to determine the area of the parallelogram that is constructed by the two sides **a** and **b.**

Matrix Operations

Add and Subtract (Matrix)

If two matrices are added or subtracted with another, those two matrices must be of the same size. The resulting vector is of the same size of **A** and **B**.

$$\mathbf{C_{mxn}} = \mathbf{A_{mxn}} \pm \mathbf{B_{mxn}}$$

Algorithm:

$$C_{ij} = A_{ij} \pm B_{ij}$$

As shown above, the addition or subtraction of two matrices can be computed by adding/subtracting the row and column index of A with the same row and column index of B.

Multiply

Matrix multiplication is an interesting operation and has several constraints, but the algorithm is simple. The constraints can be detailed below in the equation of two matrices multiplied together to form a third matrix.

$$\mathbf{C_{mxo}} = \mathbf{A_{mxn}} \pm \mathbf{B_{nxo}}$$

The constraints are now obvious. The resulting matrix C can only be formed if the number of columns of A is equal to the number of rows of B. The resulting matrix C has its number of rows equal to the rows of A and its number of columns equal to columns of B.

Algorithm:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Essentially, the row/column index of C is populated based on the sum of the product of matrix A and B with respect to the rows and columns of A and B. The sum is over the number of columns of A.

Scale (Matrix)

Scaling a matrix by a constant c is similar to the scaling of a vector. Each entry of the matrix is multiplied by the constant c.

$$\mathbf{A_{mxn}} = c(\mathbf{A_{mxn}})$$

Algorithm:

$$A_{ij} = c * A_{ij}$$

MaxNorm

As demonstrated in the MaxNorm operation for vectors, the maximum norm of a matrix is similar. The result is a scalar that is the maximum absolute value of all the entries of the matrix.

$$max\ norm = \|\mathbf{A}\|_{max}$$

Algorithm:

$$max\ norm = abs(A_{1,1})$$

$$iteration\ max\ norm = abs(A_{i,j})$$

$$If\ \text{iteration max norm} \geq \text{max norm}\ ,\ then\ \text{max norm} = \text{iteration max norm}$$

Transpose

The transpose operation creates a new matrix, that of which contains entries of the rows equal to the entries of the columns of the original. Therefore, the size of the new matrix is opposite to that of the original matrix.

$$\mathbf{B}_{nxm} = \mathbf{A}^{T}_{mxn}$$

Algorithm:

$$B_{ji} = A_{ij}$$

MatMultVec (Multiplication of Matrix by Vector)

The multiplication of a matrix by a vector has several constraints that prevent the operation from computing successfully. In order for matrix-vector multiplication to take place, the number of columns of the matrix must be equal to the number of rows of the vector. The result is a vector of size equal to the number of rows of the matrix. The equation is shown below.

$$\mathbf{b}_{mx1} = \mathbf{A}_{mxn}\mathbf{x}_{nx1}$$

Algorithm:

$$b_i = \sum_{j=1}^{n} A_{ij}x_j$$

This operation is the essential foundation for solving multiple linear algebraic equations.

AxEqb (Gaussian Elimination)

One method that can be used to solve linear algebraic equations in the form of $\mathbf{Ax} = \mathbf{b}$ is Gaussian Elimination. Gaussian Elimination utilizes forward and backward substitution to solve multiple linear algebraic equations. Forward substitution is composed of manipulating the matrix A into an equivalent matrix by utilizing three legal operations:

1. Interchange the order of two equations
2. Both sides of an equation may be multiplied by a non-zero constant
3. A multiple of one equation can be added to another equation

Within the confines of forward substitution, and equivalent matrix is generated in the form of an upper triangular matrix. Stepping throughout the equations that make up the equivalent matrix, each unknown is eliminated until only one unknown remains. This ends the forward substitution phase. The backward substitution phase then begins, which starts by solving for the last unknown in the forward substitution phase, and then subsequently solves for the next unknown using the previously solved for unknown(s).

Forward Substitution,

Looping through $k = 1 \ldots n-1$ where n is the number of rows,

$$A_{ij}^{(k)} = A_{ij}^{(k-1)} - \frac{A_{ik}^{(k-1)}}{A_{kk}^{(k-1)}}A_{kj}^{(k-1)}$$

$$b_i^{(k)} = b_i^{(k-1)} - \frac{A_{ik}^{(k-1)}}{A_{kk}^{(k-1)}}b_k^{(k-1)}$$

Where $i, j = k+1, .., n$.

We must note here that if $abs\left(A_{kk}^{(k-1)}\right) \leq \epsilon$ where ϵ is a small positive constant, then the set of equations is linearly dependent. If the result is larger than the constant, it is assumed the set of equations is linearly independent and therefore the only solution to the set of equations is the trivial solution ($\mathbf{x} = \mathbf{0}$)

Backward Substitution,

Solving for the last unknown,

$$x_n = \frac{b_n}{A_{nn}}$$

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} A_{ij} x_j}{A_{ii}}$$

Where $i = n-1, n-2, .., 1$. This ends the backward substitution phase, and the solution is now complete.

LDLT Factorization and LDLT Solve

Similar to Gaussian Elimination, LDLT Factorization and Solve is another method used to solve linear algebraic equations. The theory is that a square, symmetric and positive definite matrix can be decomposed into a lower triangular and diagonal matrix. In essence, the characteristics of this decomposed matrix can be taken advantage of in solving linear algebraic equations. Then, forward and backward substitution (similar to Gaussian) can be utilized to solve for the unknowns.

$$\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^{\mathbf{T}}$$

Note that for this implementation, the matrix A is simply overwritten with the lower triangular and diagonal matrices since the forward and backward substitution phases only require specific elements that are already stored in the respected storage locations. The equations used to perform the subsequent forward and backward substitutions are shown below.

$$\mathbf{L}\mathbf{D}\mathbf{L}^{\mathbf{T}}\mathbf{x} = \mathbf{b}$$

$$Let\ \mathbf{L}\mathbf{y} = \mathbf{b}$$

$$then\ \mathbf{D}\mathbf{L}^{\mathbf{T}}\mathbf{x} = \mathbf{y}$$

Since the matrix A contains both L and the product DL^T, A can be used instead!

Algorithm:

Decomposing,

Looping through $i = 1, \ldots, n$ where n is the number of rows,

$$A_{ii} = A_{ii} - \sum_{j=1}^{i-1} A_{ij}^2 A_{jj} \; if \; A_{ii} < \epsilon, stop \; (not \; positive \; definte)$$

For $j = i + 1, \dots, n, set \; A_{ji} = \frac{A_{ji} - \sum_{k=1}^{i-1} A_{jk} A_{kk} A_{ik}}{A_{ii}}$

End loop i. This ends the decomposition of A. Now, the decomposed matrix can be used to forward substitute and then solve for the unknowns.

Forward substitution,

$$Set \; x_1 = b_1$$

For $i = 2, \dots, n, set \; x_i = b_i - \sum_{j=1}^{i-1} A_{ij} x_j$

This ends the forward substitution phase.

Backward substitution,

$$Set \; x_n = \frac{x_n}{A_{nn}}$$

For $i = n - 1, n - 2, \dots, 1, set \; x_i = \frac{x_i}{A_{ii}} - \sum_{j=i+1}^{n} A_{ji} x_j$

This ends the forward substitution phase, and the solution is complete.

Matrix Toolbox Test Cases

In order to have confidence in using the above operations to perform various tasks, these operations should adhere to several testing cases. The main purpose of testing these functions/operations is to determine whether or not these functions output an accurate result, and if the error is caught and reported.

Vector Test Cases

Test Vector 1

$$\mathbf{a} = \begin{Bmatrix} -1 \\ -2 \\ -3 \end{Bmatrix} \; and \; \mathbf{b} = \begin{Bmatrix} 1 \\ 4 \\ 9 \end{Bmatrix}$$

Result:

Function	*OUTPUT*	*Result*
Add(A and B)	<0,2,6>	Successful
Subtract(A and B)	<-2,-6,-12>	Successful
DotProduct(A and B)	-36	Successful
Normalize(A)	<-0.267261,-0.534522,-0.801784>	Successful
Scale (constant = -2) (A)	<2,4,6>	Successful
Max(A)	-1	Successful
Min(A)	-3	Successful
TwoNorm(A)	3.74166	Successful
MaxNorm(A)	3	Successful
CrossProduct(A and B)	<-6,6,-2>	Successful

Test Vector 2

$$\mathbf{a} = \begin{Bmatrix} 1 \\ -2 \end{Bmatrix} \; and \; \mathbf{b} = \begin{Bmatrix} 1 \\ 4 \\ 9 \end{Bmatrix}$$

Result:

Function	*OUTPUT*	*Result*
Add(A and B)	Error	Failed
Subtract(A and B)	Error	Failed
DotProduct(A and B)	Error	Failed
Normalize(A)	<0.447214,-0.894427>	Successful
Scale (constant = -2) (A)	<-2,4>	Successful
Max(A)	1	Successful
Min(A)	-2	Successful
TwoNorm(A)	2.23607	Successful
MaxNorm(A)	2	Successful

CrossProduct(A and B)	Error	Failed

As expected, the add, subtract and dot product functions could not compute as the size between A and B are not the same. The cross product function could not compute since A was not three-dimensional. We can also check a null vector to see if the normalize function produces an error (division of zero).

Test Vector 3

$$\mathbf{a} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \; and \; \mathbf{b} = \begin{Bmatrix} 1 \\ 4 \\ 9 \end{Bmatrix}$$

Result:

Function	*OUTPUT*	*Result*
Add(A and B)	Error	Failed
Subtract(A and B)	Error	Failed
DotProduct(A and B)	Error	Failed
Normalize(A)	Error	Failed
Scale (constant = -2) (A)	<-0,0>	Successful
Max(A)	0	Successful
Min(A)	0	Successful
TwoNorm(A)	0	Successful
MaxNorm(A)	0	Successful
CrossProduct(A and B)	Error	Failed

The result is as expected.

Matrix Test Cases

Test Matrix 1

$$\mathbf{A_{2x2}} = \begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix} \; and \; \mathbf{B_{2x2}} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Result:

Function	*OUTPUT*	*Result*
Add(A and B)	[2 4; 2 4]	Successful
Subtract(A and B)	[2 2; 4 4]	Successful
Multiply(A and B)	[-3 2; -4 3]	Successful
Scale (constant = -2) (A)	[-4 -6; -6 -8]	Successful
MaxNorm(A)	4	Successful
Transpose(A)	[2 3; 3 4]	Successful

Changing the size of the two matrices so that they are no longer equal,

Test Matrix 2

$$\mathbf{A_{3x2}} = \begin{bmatrix} 2 & 3 \\ 3 & 4 \\ 4 & 5 \end{bmatrix} and\ \mathbf{B_{2x2}} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Result:

Function	*OUTPUT*	*Result*
Add(A and B)	Error	Failed
Subtract(A and B)	Error	Failed
Multiply(A and B)	[-3 2; -4 3; -5 4]	Successful
Scale (constant = 1) (A)	[2 3; 3 4; 4 5]	Successful
MaxNorm(A)	5	Successful
Transpose(A)	[2 3 4; 3 4 5]	Successful

Add and subtract do not compute, since the size of A and B are not equal. Matrix multiplication does work, since the columns of A are equal to the rows of B. Changing the size of B,

Test Matrix 3

$$\mathbf{A_{3x2}} = \begin{bmatrix} 2 & 3 \\ 3 & 4 \\ 4 & 5 \end{bmatrix} and\ \mathbf{B_{3x3}} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & -2 \\ 4 & 2 & 5 \end{bmatrix}$$

Function	*OUTPUT*	*Result*
Add(A and B)	Error	Failed
Subtract(A and B)	Error	Failed
Multiply(A and B)	Error	Failed

Multiplication fails since the columns of A do not match the rows of B. Testing the matrix-vector multiplication function.

$$\mathbf{A_{3x2}} = \begin{bmatrix} 2 & 3 \\ 3 & 4 \\ 4 & 5 \end{bmatrix} and\ \mathbf{x_{2x1}} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Since the columns of A match the rows of x, multiplication is valid.

$$b = \begin{bmatrix} 8 \\ 11 \\ 14 \end{bmatrix}$$

Conversely, if the columns do not match the rows, an error protrudes. We can also test LDLT factorization, as well. Since the requirement is that the matrix A is square, symmetric and positive definite, we can try all three cases and see if the toolbox provides an error message. Using valid values for LDLT Factorization and Solve, we get the following decomposed matrix A (using the legal matrix provided in Example 10.2.4 of the E-Book).

```
LDLT Factorization of Matrix A .....

Decomposed Matrix A
(1,1) 3.512             (1,2) 0.7679            (1,3) 0                 (1,4) 0

(1,5) 0
(2,1) 0.21865           (2,2) 2.9841            (2,3) 0                 (2,4) -2

(2,5) 0
(3,1) 0                 (3,2) 0                 (3,3) 3.512             (3,4) -0.76
79
(3,5) 0.7679
(4,1) 0                 (4,2) -0.670219         (4,3) -0.21865          (4,4) 1.643
66
(4,5) -1.152
(5,1) 0                 (5,2) 0                 (5,3) 0.21865           (5,4) -0.59
8724
(5,5) 2.3949


LDLT Solve of Ax = b .....

Vector x in Ax = b
(1) 0.00444367      (2) -0.0203232      (3) -0.00444367      (4) -0.0303232

(5) -0.01
Residual of Ax - b = 0.0176488
```

The decomposed matrix will only decompose if the matrix A is square, symmetric and positive definite. If any of those constraints are invoked, the program will produce an error. The LDLT Solve function will then perform forward and back substitution to solve for the linear algebraic equations. Computing the residual and taking the magnitude of the residual,

$$\|r = Ax - b\| = 0.0176488$$

We can compare this to the residual magnitude from Gaussian Elimination (of the same algebraic equations), which is,

$$\|r\| = 0.0442601$$

As we can see from the residuals, LDLT produces slightly more favorable results.