

6.2 Working with Matrices

We have already defined both vectors and matrices and discussed how to create them in the *Vectors and Matrices in MATLAB* section.

6.2.1 Special Matrices

Zero Matrix

A **zero** vector or matrix of any size with all zero elements is denoted as **0**.

Diagonal Matrix

A **diagonal** matrix has zeros everywhere except on the main diagonal, which is the set of elements where row index and column index are the same. Diagonal matrices are usually square (same number of rows and columns), but they may be rectangular.

$$A = \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} \quad \begin{array}{l} v = \text{diag}(A) \\ A = \text{diag}(v) \end{array}$$

The *main* diagonal, also called the *forward* diagonal, or the *major diagonal*, is the set of diagonal matrix elements from upper left to lower right. The set of diagonal elements from lower left to upper right is of significantly less interest to us, but has several names including the *antidiagonal*, *back diagonal*, *secondary diagonal*, or the *minor diagonal*.

Identity Matrix

An **identity** matrix (I) is a square, diagonal matrix where all of the elements on the main diagonal are one. Identity matrices are like a one in scalar math. That is, the product of any matrix with the identity matrix yields itself.

$$\underline{AI} = A = \underline{IA}$$

$$I_{2 \times 2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$I_{3 \times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$I_{4 \times 4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Handwritten example showing matrix multiplication:

$$\begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix}$$

MATLAB has a function called eye that takes one argument for the matrix size and returns an identity matrix.

```
>> I2 = eye(2)
I2 =
    1    0
    0    1
>> I3 = eye(3)
I3 =
    1    0    0
    0    1    0
    0    0    1
```

Upper Triangular Matrix

Upper triangular matrices have all zero values below the main diagonal. Any non-zero elements are on or above the main diagonal.

$$U = \begin{bmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{bmatrix}$$

Lower Triangular Matrix

Lower triangular matrices have all zero values above the main diagonal. Any non-zero elements are on or below the main diagonal.

$$L = \begin{bmatrix} a & 0 & 0 \\ b & c & 0 \\ d & e & f \end{bmatrix}$$

Symmetric Matrix

A **symmetric** matrix (S) has symmetry relative to the main diagonal. If the matrix were written on a piece of paper and you folded the paper along the forward diagonal then the off-diagonal elements with the same value would lie on top of each other. Thus for symmetric matrices $S^T = S$. Here are a couple example symmetric matrices.

$$\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \quad \begin{bmatrix} 2 & 3 & 6 \\ 3 & 5 & 5 \\ 6 & 5 & 2 \end{bmatrix}$$

Orthonormal Matrix

A matrix is called *orthonormal* if the columns are unit vectors (length of 1) and the dot product between the columns is zero ($\cos(\theta) = 0$). That is to say, the columns are all orthogonal to each other.

Orthogonal Matrix

A matrix is **orthogonal** (Q) if it is both orthonormal and square.

6.2.2 Special Matrix Relationships

Matrix Inverse

The inverse of a square matrix, A , is another matrix, A^{-1} , that multiplies with the original matrix to yield the identity matrix.

$$A^{-1}A = AA^{-1} = I$$

Not all square matrices have an inverse and calculating the inverse, especially for larger matrices is a nontrivial, which will be discussed later.

In MATLAB, the function `inv(A)` returns inverse of matrix A.

Here are a couple simple properties for matrix inverses.

- $(AB)^{-1} = B^{-1}A^{-1}$ This is a simple proof that is discussed in one of the attendance quizzes. Hint: Start with $(AB)^{-1}(AB) = I$.
 $(AB)^{-1}AB = I \implies \underbrace{(AB)^{-1}A} B = I \implies (AB)^{-1}A = B^{-1} \implies (AB)^{-1} = B^{-1}A^{-1}$
- $(A^T)^{-1} = (A^{-1})^T$.
 Start with the simple observation that $I^T = I$ and $(AB)^T = B^T A^T$ (see *Matrix Transpose Properties*).
 $AB(A^T)^{-1} = I$

$$\begin{aligned} (A^{-1}A)^T &= I^T = I \\ A^T(A^{-1})^T &= I \\ \underline{(A^T)^{-1}A^T(A^{-1})^T} &= \underline{(A^T)^{-1}} \\ \underline{(A^{-1})^T} &= \underline{(A^T)^{-1}} \end{aligned}$$

Matrix Transpose Properties

We described the transpose for vector and matrices in *Transpose*.

The [Wikipedia page for transpose](#) lists many transpose properties.

- $(A^T)^T = A$
- The transpose with respect to addition, $(A + B)^T = A^T + B^T$.
- $(AB)^T = B^T A^T$. Notice that the order is reversed.

$$\begin{aligned} (i, j) \text{ value of } \underline{(AB)^T} &= (j, i) \text{ value of } (AB) \\ &= j \text{ row of } A \times i \text{ column of } B \\ &= i \text{ row of } B^T \times j \text{ column of } A^T \\ &= (i, j) \text{ value of } \underline{(B^T A^T)} \end{aligned}$$

- We often pre-multiply vectors and matrices by their transpose ($A^T A$). The result is a scalar for a column vector, and a square, symmetric matrix for a row vector, rectangular matrix, and square matrix. The operation of $A^T A$ turns out to be a very useful operation for rectangular matrices. For a m-by-n matrix, the resulting matrix size is n-by-n (square).

If $S = \underline{A^T A}$ is symmetric, then $\underline{S^T = S}$.

$$\underline{S^T} = (\underline{A^T A})^T = A^T (A^T)^T = A^T A = \underline{S}$$

Another interesting result is that the trace of $S = A^T A$, denoted as $Tr(S)$, is the sum of the squares of all the elements of A . The trace of a matrix is the sum of its diagonal elements.

$$A^T A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix} = \begin{bmatrix} a^2 + b^2 + c^2 & ad + be + cf \\ ad + be + cf & d^2 + e^2 + f^2 \end{bmatrix}$$

- For an orthogonal matrix, $Q^{-1} = Q^T$. This is called the *Spectral Theorem*. Note that the columns of the matrix must be unit vectors for this property to be true.

Since each of the columns are unit vectors, it follows that the diagonal of $Q^T Q$ is ones. The off-diagonal values are zeros because the dot product between the columns is zero. Thus,

$$Q^T Q = Q Q^T = I$$

$$Q^T = Q^{-1}$$

Here is an example using an orthogonal matrix, which is a rotation matrix. Rotation matrices are often used in engineering.

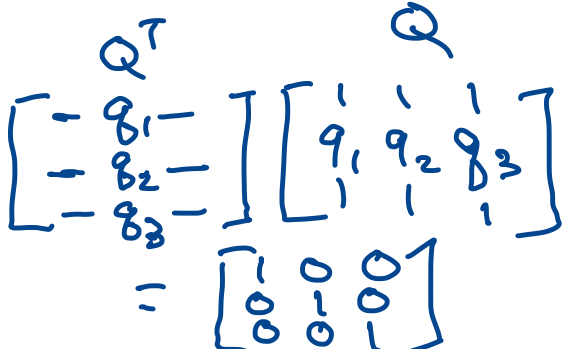
```
>> Q
Q =
    0.5000    -0.8660
    0.8660     0.5000

>> Q'
ans =
    0.5000     0.8660
   -0.8660     0.5000

>> Q(:,1)'*Q(:,2)    % dot product of columns => orthogonal
ans =
    0

>> Q'*Q              % orthogonal columns
ans =
    1.0000    -0.0000
   -0.0000     1.0000

>> Q*Q'              % orthogonal rows
ans =
    1.0000     0.0000
    0.0000     1.0000
```



Another interesting result about orthogonal matrices is that the product of two orthogonal matrices is also orthogonal. This comes about because both the inverse and transpose of matrices reverse the matrix order. Proof of orthogonality only requires that $Q^T = Q^{-1}$.

Consider two orthogonal matrices Q_1 and Q_2 .

$$(Q_1 Q_2)^{-1} = Q_2^{-1} Q_1^{-1} = Q_2^T Q_1^T = (Q_1 Q_2)^T$$

6.2.3 Matrix Math

Addition and Subtraction

Addition and subtraction of matrices is performed element-wise.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} - \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a - e & b - f \\ c - g & d - h \end{bmatrix}$$

Scalar and element-wise operations between matrices work the same as with vectors. (See *Element-wise Arithmetic*)

Matrix Multiplication

Matrix multiplication requires that the inner dimensions of the two matrices agree. A m-by-n matrix may be multiplied by a n-by-p matrix to yield a m-by-p matrix. The number of columns in the first matrix, n , must equal the number of rows in the second matrix, n .

Multiplication of a matrix by a vector is defined as either the linear combination of the columns of the matrix with the vector elements, or as the sum of products between the rows of left matrix and the columns of right matrix or vector.

$$\begin{aligned} Ax &= \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 2 \\ 6 & -3 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} \\ &= 2 \begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix} + 1 \begin{bmatrix} 2 \\ 5 \\ -3 \end{bmatrix} + 1 \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} [1 \ 2 \ 3] \cdot [2 \ 1 \ 1] \\ [2 \ 5 \ 2] \cdot [2 \ 1 \ 1] \\ [6 \ -3 \ 1] \cdot [2 \ 1 \ 1] \end{bmatrix} \\ &= \begin{bmatrix} 7 \\ 11 \\ 10 \end{bmatrix} \end{aligned}$$

column view

row view
Inner product

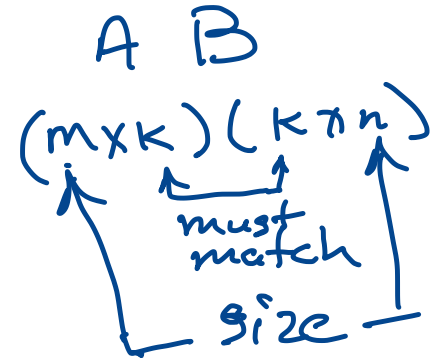
Each element of a matrix multiplication between two matrices comes from the sum of products between a row of the first matrix by a column of the second matrix.

The product C of matrices A (M rows and K columns) and B (K rows and N columns) is a matrix of M rows and N columns.

$$C = AB$$

The individual values of $C_{i,j}$ are calculated as:

$$C_{i,j} = \sum_{n=1}^K A_{i,n} B_{n,j}$$



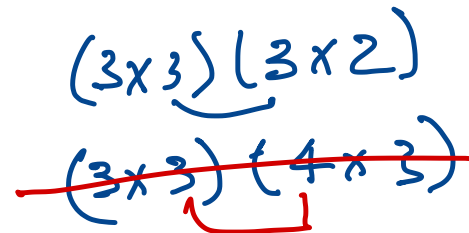
For 2-by-2 matrices:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} (ae + bg) & (af + bh) \\ (ce + dg) & (cf + dh) \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} (1 \cdot 5 + 2 \cdot 7) & (1 \cdot 6 + 2 \cdot 8) \\ (3 \cdot 5 + 4 \cdot 7) & (3 \cdot 6 + 4 \cdot 8) \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

Table 6.1: Size of Matrix Multiplications

First Matrix	Second Matrix	Output Matrix
1-by- n row vector	n -by-1 column vector	1-by-1 scalar
1-by- p row vector	p -by- n matrix	1-by- n row vector
n -by-1 column vector	1-by- m row vector	n -by- m matrix
n -by- p matrix	p -by-1 column vector	n -by-1 column vector



Matrix Multiplication Properties

1. Matrix multiplication is **associative**: $(AB)C = A(BC)$.
2. Matrix multiplication is, in general, **NOT commutative** even for square matrices: $(AB \neq BA)$. The exceptions to this are multiplication by the identity matrix and inverse matrix.
3. $AB = AC$ does **not** necessarily imply that $B = C$. For example, consider matrices:

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}, B = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}, C = \begin{bmatrix} 4 & 3 \\ 0 & 2 \end{bmatrix}$$

The Outer Product View of Matrix Multiplication

We normally think of matrix multiplication as finding each term of the matrix product from the sum of products between the rows of the first matrix and the columns of the second matrix. This

is the *inner product* or dot product view. But there is also an outer product view that is columns times rows.

$$AB = \begin{bmatrix} | & | & \cdots & | \\ a_1 & a_2 & \cdots & a_n \\ | & | & \cdots & | \end{bmatrix} \begin{bmatrix} - & b_1 & - \\ - & b_2 & - \\ \vdots & \vdots & \vdots \\ - & b_n & - \end{bmatrix} = \underline{a_1 b_1} + \cdots + a_n b_n$$

Each column a_k of an m-by-p matrix multiplies the k th row of b_k of a p-by-n matrix. The product $a_k b_k$ is an m-by-n matrix of *rank one* formed from the *Outer Product* of the two vectors. The final product AB is the sum of the rank-one matrices.

The outer product matrix multiplication is not, in practice, how to multiply two matrices. Rather, it gives an alternative perspective of what matrix multiplication accomplishes. This perspective will be useful later when we learn about various ways to factor a matrix into a product of sub-matrices, particularly with the *Singular Value Decomposition (SVD)*.

Now complete *A Change of Coordinate Frames* homework assignment.

Matrix Division

Matrix division, in the sense of scalar division, is not defined for matrices. Multiplication by the inverse of a matrix accomplishes the same objective as division does in scalar arithmetic. MATLAB also has two alternative operators to the matrix inverse that act as one might expect from division operators, left-divide (\backslash) and right-divide ($/$); however, they are much more complicated than simple division. They are used to solve *Systems of Linear Equations*.

Table 6.2: MATLAB's Matrix Division Operators

Operator	Need	Usage	Replaces	Common Name
\backslash	$A*x = b$	$x = A \backslash b$	$x = \text{inv}(A)*b$	A under b
$/$	$x*A = b$	$x = b / A$	$x = b*\text{inv}(A)$	b over A

$$\begin{aligned}
 ax &= b \\
 x &= \frac{b}{a} \\
 x &= A^{-1}b \\
 \cancel{A^{-1}A}x &= A^{-1}b
 \end{aligned}$$

6.2.4 Determinant

The determinant of a square matrix is a number. At one time, a significant portion of linear algebra centered around determinants, but that is not so much the case today [STRANG99]. Thus, while it is important to know what a determinant is, we will avoid using them when possible. I'm not aware of any algorithms internal to MATLAB that compute determinants of matrices.

- The complexity of computing a determinant by the normal method has run-time complexity on the order of $n!$ (n – factorial), which is fine for small matrices, but is too slow for large matrices. Although, there is a *Determinant Shortcut* using *LU Decomposition* that improves it to $O(n^3)$.

$$O(n!)$$

- With the use of computers and applications such as machine learning that use linear algebra significantly, very large matrices are often used, so computing determinants is less useful because it is possible to use alternative algorithms that do not require determinants. New algorithms for applications that previously used determinants have been found that do not require determinants. This is the case for:
 - determining if a matrix is invertible.
 - computing eigenvalues
- One application of the determinant is to compute the volume of a parallelepiped. A *parallelepiped* is a generalization of a cube, it is a three-dimensional figure formed by six parallelograms. The absolute value of the determinant of real vectors is equal to the volume of the parallelepiped spanned by those vectors.

Either of two notations are used to indicate the determinant of a matrix – $\det(\mathbf{A})$ or $|\mathbf{A}|$.

The standard technique for computing a determinant is called the **Laplace expansion**.

The determinant of a 2-by-2 matrix is simple, but it gets more complicated as the matrix dimension increases.

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - cb$$

The 3-by-3 determinant makes use of 3 2-by-2 determinants, called cofactors. Each element in a row or column is multiplied by a determinant from the other rows and columns excluding the row and column of the multiplication element.

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

Another approach to remembering the sum of products need to compute a 3x3 determinant is to compute diagonal product terms. The terms going from left to right are added while the right to left terms are subtracted. Note: This only works for 3x3 determinants.

$$\begin{pmatrix} a & b & c & a & b \\ d & e & f & d & e \\ g & h & i & g & h \end{pmatrix} - \begin{pmatrix} a & b & c & a & b \\ d & e & f & d & e \\ g & h & i & g & h \end{pmatrix}$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = aei + bfg + cdh - ceg - afh - bdi$$

For a 4-by-4 determinant, the pattern continues with each cofactor term now being a 3-by-3 determinant. The pattern likewise continues for higher order matrices. The number of computations

needed with this method of computing a determinant is on the order of $n!$, which is prohibitive for large matrices.

Notice that the sign of the cofactor additions alternate according to the pattern:

$$sign_{i,j} = (-1)^{i+j} = \begin{bmatrix} + & - & + & - \\ - & + & - & + \\ + & - & + & - \\ - & + & - & + \end{bmatrix}$$

It is not necessary to always use the top row for Laplace expansion. Any row or column will work, just make note of the signs of the cofactors. The best strategy is to expand along the row or column with the most zeros.

Here is bigger problem with lots of zeros so it is not too bad, so try using Laplace expansion to find the determinant.

$$\begin{array}{c} + \\ - \\ + \\ - \end{array} \left| \begin{array}{ccccc} 8 & 5 & 4 & 3 & 0 \\ 7 & 0 & 6 & 1 & 0 \\ 8 & -5 & 4 & 3 & -5 \\ -3 & 0 & 0 & 0 & 0 \\ 4 & 0 & 2 & 2 & 0 \end{array} \right|$$

MATLAB has a function called `det` that takes a square matrix as input and returns the determinant.

$$\begin{array}{c} + \quad - \quad + \quad - \\ 3 \left| \begin{array}{cccc} 5 & 4 & 3 & 0 \\ 0 & 6 & 1 & 0 \\ 5 & 4 & 3 & 5 \\ 0 & 2 & 2 & 0 \end{array} \right| \end{array} = (3)(5) \left| \begin{array}{ccc} 6 & 1 & 0 \\ 0 & 6 & 1 \\ 0 & 2 & 2 \end{array} \right|$$

$$= 15 \cdot 5 \left| \begin{array}{cc} 6 & 1 \\ 2 & 2 \end{array} \right| = 15 \cdot 5 (12 - 2)$$

$$= 15 \cdot 5 \cdot 10$$

$$= \underline{\underline{750}}$$

6.2.5 Calculating a Matrix Inverse

The inverse of a square matrix, A , is another matrix, A^{-1} that multiplies with the original matrix to yield the identity matrix.

$$A^{-1}A = AA^{-1} = I$$

Unfortunately, calculating the inverse of a matrix is not a trivial problem. A formula known as *Cramer's Rule* provides a neat and tidy equation for the inverse, but for matrices beyond a 2-by-2, it is computationally very slow. Cramer's Rule requires the calculation of the determinant and the full set of covariances for the matrix. The *run time complexity of Cramer's rule is $O(n \cdot n!)$* , which means that for a n-by-n matrix, calculating an inverse using Cramer's rule requires about $n \cdot n!$ computations. The covariances of a 2-by-2 matrix are simple, which is not true for larger matrices.

For a 2-by-2 matrix, Cramer's rule can be found by algebraic substitution.

$$AA^{-1} = I$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

One can perform the above matrix multiplication and find four simple equations from which the terms of the matrix inverse may be derived.

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \text{ has inverse } A^{-1} = \frac{\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}}{\det(A)} = \frac{1}{(ad - cd)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Because of the computational complexity, Cramer's Rule is not used by MATLAB or similar software. Another technique known as *Elimination* is used. Implementing the elimination technique in software is more complex, but the result is much faster calculations. The *run time complexity of calculating a matrix inverse by elimination is $O(n^3)$* . The good news is that MATLAB already has a matrix inverse function called `inv`.

```
>> A = [2 -3 0;4 -5 1;2 -1 -3];
>> A_inv = inv(A)
A_inv =
    -1.6000    0.9000    0.3000
    -1.4000    0.6000    0.2000
    -0.6000    0.4000   -0.2000
```

Note: MATLAB's *left-divide operator* uses the *Elimination* technique to solve a system of equations after only computing approximately half of the inverse of the matrix. Thus, it is not usually necessary to compute the full inverse of the matrix.

6.2.6 Invertible Test

Not all matrices can be inverted. A matrix that is not invertible is said to be *singular*. A matrix is singular if some of its rows or columns are dependent on each other. An invertible matrix must derive from the same number of independent equations as the number of unknown variables. A square matrix from independent equations has *rank* equal to the dimension of the matrix, which means it is *full rank*, has a non-zero determinant, and is invertible. We will define *rank* more completely after discussing *Elimination*. For now, just think of *rank* as the number of independent equations represented by a matrix.

The following matrix is singular because column 3 is just column 1 multiplied by 2.

$$\begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 4 \\ 1 & 2 & 2 \end{bmatrix}$$

Sometimes it is difficult to observe that the rows or columns are not independent. When a matrix is singular, its rank is less than the dimension of the matrix, and its *Determinant* also evaluates to zero.

```
>> A = [1 0 2;2 1 4;1 2 2];
>> rank(A)
ans =
     2
>> det(A)
ans =
     0
>> det(A') % the transpose is also singular
ans =
     0
```

If you work much with MATLAB, you will occasionally run across an error or warning message saying that a matrix is close to singular. The message may also reference a condition or rcondition number. The condition number is another test for invertibility, which we will describe when we get to *Singular Value Decomposition (SVD)*.

We will mostly use rank as our invertible test.

6.2.7 Cross Product

The cross product is a special operation using two vectors in \mathbb{R}^3 with application to geometry and physical systems. Although it is an operation for vectors, it is included here because matrix operations (determinant or multiplication) is used to compute it.

The cross product of two non-parallel 3-D vectors is a vector perpendicular to both vectors and the plane which they span. Finding the perpendicular to a plane is used to write the equation of a

plane. The *Projections Onto a Hyperplane* has an example of using a cross product in the equation of a plane. The direction of the cross product vector is determined by the right hand rule.

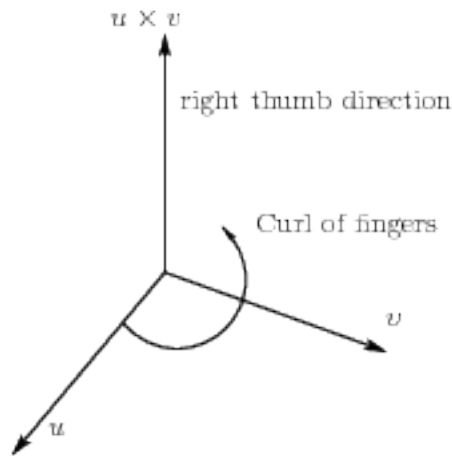


Fig. 6.2: **Right Hand Rule:** $\mathbf{u} \times \mathbf{v}$ points along your right thumb when the fingers curl from \mathbf{u} to \mathbf{v} .

- $\mathbf{u} \times \mathbf{v}$ is spoken as “**u** cross **v**”.
- $\mathbf{u} \times \mathbf{v}$ is perpendicular to both \mathbf{u} and \mathbf{v} .
- $\mathbf{v} \times \mathbf{u}$ is $-(\mathbf{u} \times \mathbf{v})$.
- The length $\|\mathbf{v} \times \mathbf{u}\| = \|\mathbf{u}\| \|\mathbf{v}\| |\sin \theta|$, where θ is the angle between \mathbf{u} and \mathbf{v} .
- The MATLAB function `cross(u, v)` returns $\mathbf{u} \times \mathbf{v}$.

Cross Product by Determinant

The cross product of $\mathbf{u} = (u_1, u_2, u_3)$ and $\mathbf{v} = (v_1, v_2, v_3)$ is a vector.

$$\mathbf{u} \times \mathbf{v} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{vmatrix} = \vec{i}(u_2v_3 - u_3v_2) - \vec{j}(u_1v_3 - u_3v_1) + \vec{k}(u_1v_2 - u_2v_1)$$

Vectors \vec{i} , \vec{j} , and \vec{k} are the unit vectors in the directions of x , y , and z of the 3-D axis.

Cross Product by Skew-Symmetric Multiplication

An alternative way to compute $\mathbf{u} \times \mathbf{v}$ is by multiplication of a *skew-symmetric*, or *anti-symmetric* matrix.

- The skew-symmetric matrix of \mathbf{u} is given the math symbol, $[\mathbf{u}]_{\times}$. Such a matrix has a zero diagonal and is always singular. The transpose of a skew-symmetric matrix is equal to its negative.

$$[\mathbf{u}]_{\times}^T = -[\mathbf{u}]_{\times}$$

•MATLAB function `skew(u)` returns $[\mathbf{u}]_{\times}$.

•For vectors in \mathbb{R}^3 :

$$[\mathbf{u}]_{\times} = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix}$$

$$\mathbf{u} \times \mathbf{v} = [\mathbf{u}]_{\times} \mathbf{v}$$

```
>> u = [1 2 3]';
>> v = [1 0 1]';
>> cross(u,v)
ans =
     2
     2
    -2
>> s = skew(u)
s =
     0     -3     2
     3     0    -1
    -2     1     0
>> s*v
ans =
     2
     2
    -2
```