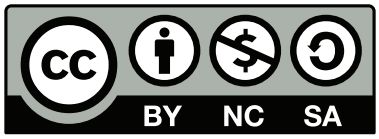
Aerospace Simulation

Tamas Kis | tamas.a.kis@outlook.com

Tamas Kis
https://tamaskis.github.io



Contents

II Modeling the Environment

III Appendices

List of Algorithms

Angles

Rotations

Kinematics

Time Units

Time Scales

Preface

THE primary goal of this text is to bridge the gap between theory and implementation for aerospace simulations. The end result of all discussions is an algorithm (or a set of algorithms) that presents a clear computation procedure that can be implemented in any programming language. A lot of theory behind the algorithms *is* covered, but I choose to focus more on developing procedures that can be easily implemented, and less on rigorously explaining every little mathematical detail.

Notation

a	scalar
$\mathbf{a}$	column vector
$\boldsymbol{a}$	physical vector
${}^{\mathrm{cf}}\boldsymbol{a}$	physical rate vector relative to the CF coordinate frame
$\dot{\mathbf{a}}$	time derivative of the vector $\mathbf{a}$ TODO
$\mathbf{a}_{\mathrm{cf}}$	coordinate vector of $\boldsymbol{a}$ expressed in the CF coordinate frame
$\mathbf{a}$	vector resolved in principal frame
$\hat{\mathbf{a}}$	unit vector in the direction of the column vector $\mathbf{a}$
$\hat{\boldsymbol{a}}$	unit vector in the direction of the physical vector $\mathbf{a}$
$\mathbf{A}$	matrix
$\boldsymbol{A}$	physical matrix
$\mathbf{I}_{n\times n}$	$n \times n$ identity matrix
$\boldsymbol{I}$	inertia tensor
$\mathbf{I}$	principal inertia tensor
${}^{\mathrm{cf}}\mathbf{I}$	inertia tensor relative to the CF coordinate frame
$\mathbf{R}_{\mathrm{cf1}\rightarrow\mathrm{cf2}}$	rotation matrix that transforms the components of a vector from the CF1 coordinate frame to the CF2 coordinate frame
$\mathbf{q} = (q_0, q_1, q_2, q_3)^T$	quaternion

Important Items of Note:

1. If the "left superscript" (i.e. ${}^{\mathrm{cf}}\boldsymbol{a}$) is omitted, it is assumed the vector is defined with respect to an inertial frame.

PART I

Kinematics

1

Column Vectors

1.1 Column Vectors and Matrices

In mathematics, a **scalar** is a single number. An example of a scalar is

$$a = 5 \in \mathbb{R}$$

Convention 1: Notation for scalars.

Scalars are written using lowercase or uppercase, italic, non-boldface symbols. For example, a real-valued scalar may be written as

$$a \in \mathbb{R}$$

In mathematics, a **matrix** is a rectangular array of numbers. An example of a matrix is

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 0 & 5 \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

Convention 2: Notation for matrices.

Matrices are written using uppercase, upright, boldface symbols. For example, a real-valued matrix of size $m \times n$ may be written as

$$\mathbf{A} \in \mathbb{R}^{m\times n}$$

In mathematics, a **vector** is a list of numbers. An example of a vector is

$$\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \in \mathbb{R}^4$$

In this text, we refer to such a vector as a **column vector**. Note that it is standard to assume that all vectors are **column**

vectors. If a vector is arranged in a row, we refer to it explicitly as a **row vector**, and denote its size differently. For example, an n-dimensional real-valued row vector is denote as being a member of the vector space $\mathbb{R}^{1\times n}$. An example of a row vector is

$$\mathbf{v} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \in \mathbb{R}^{1\times 4}$$

Convention 3: Notation for column vectors.

Column vectors are written using lowercase, upright, boldface symbols. For example, a real-valued vector of length n may be written as

$$\mathbf{v} \in \mathbb{R}^n$$

Real-valued column vectors have both a magnitude and a direction. The **magnitude** of column vector $\mathbf{v}$ is calculated using the 2-norm.

$$\boxed{\|\mathbf{v}\| = v = \text{magnitude of } \mathbf{v}} \tag{1.1}$$

The **2-norm** of a column vector is defined as

$$\boxed{\|\mathbf{v}\| = \sqrt{\mathbf{v}^T\mathbf{v}}} \tag{1.2}$$

where the "T" denotes the transpose operation.

Convention 4: Notation for the magnitude of a column vector.

Since the magnitude of a column vector is a scalar, it is written using the italic, non-boldface version of the same symbol. For example, the magnitude of $\mathbf{v}$ is written as

$$v$$

To describe the **direction** of a column vector, we use unit vectors. A **unit vector** is a vector of magnitude 1. Therefore, if we multiply a scalar quantity by a unit vector, we get a vector whose magnitude is equal to the scalar and whose direction is parallel to the unit vector's. The unit vector in the direction of $\mathbf{v}$ is defined as

$$\boxed{\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|} = \frac{\mathbf{v}}{v} = \text{direction of } \mathbf{v}} \tag{1.3}$$

Convention 5: Notation for mathematical unit vectors.

A mathematical unit vector in the direction of a column vector is written using the same symbol but with a hat over it. For example, the unit vector in the direction of $\mathbf{v}$ is written as

$$\hat{\mathbf{v}}$$

By rearranging Eq. (1.3), we can write a vector $\mathbf{v}$ in terms of its magnitude and direction.

$$\boxed{\mathbf{v} = v\hat{\mathbf{v}}} \tag{1.4}$$

1.2 Coordinate Systems

A **coordinate system** is a system that uses one or more numbers (i.e. coordinates) to determine the position of some geometric element with respect to that coordinate system. We use the following convention to name coordinate systems:

Convention 6: Naming coordinate systems.

Consider a coordinate system with origin O and axes $x_1, ..., x_n$. We refer to this coordinate system as $Ox_1...x_n$.

As a visual example, consider the three-dimensional coordinate system $Ox_1x_2x_3$ is illustrated in Fig. 1.1.

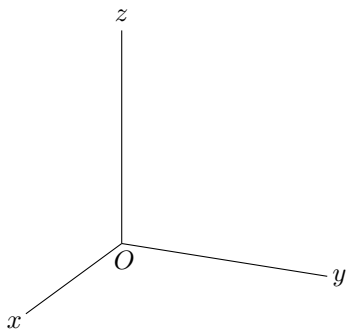


Figure 1.1: $Ox_1x_2x_3$ coordinate system.

A coordinate system is defined using an **ordered basis**, or simply a **basis** (since most bases we deal with in this text are ordered bases). Let X be the basis defining the $Ox_1...x_n$ coordinate system. Then

$$\boxed{X = (\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_n) = \text{ordered basis}} \tag{1.5}$$

Convention 7: Ordered basis notation.

An ordered basis is denoted using an uppercase, italic character, for example

$$X$$

$\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_n$ are referred to as **basis vectors** since they define an ordered basis. In this text, we assume that all basis vectors are unit vectors, i.e. they all a of have magnitude 1[1].

$$\boxed{\|\hat{\mathbf{x}}_1\| = \cdots = \|\hat{\mathbf{x}}_n\| = 1} \tag{1.6}$$

Also note that all basis vectors have dimension equal to the dimension of the coordinate system they collectively define. For an n-dimensional coordinate system [24][26, pp. 157–162],

$$\boxed{\hat{\mathbf{x}}_i \in \mathbb{R}^n \quad \forall\, i = 1, ..., n} \tag{1.7}$$

[1] This is not required of basis vectors, but in the context of this text we make this assumption since most bases we will deal with have unit basis vectors

In our three-dimensional example, the tails of $\hat{\mathbf{x}}_1$, $\hat{\mathbf{x}}_2$, and $\hat{\mathbf{x}}_3$ are all located at the origin O, while their heads points in the directions of the x_1, x_2, and x_3 axes, respectively. The $Ox_1x_2x_3$ coordinate system together, with its basis vectors $\hat{\mathbf{x}}_1$, $\hat{\mathbf{x}}_2$, and $\hat{\mathbf{x}}_3$, is depicted in Fig. 1.2.

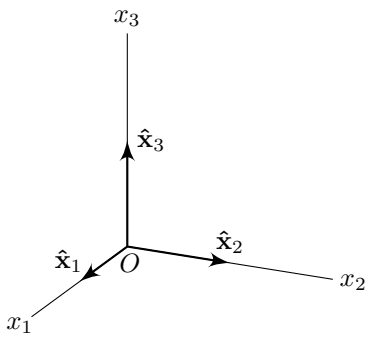


Figure 1.2: $Ox_1x_2x_3$ coordinate system with basis vectors.

1.3 Coordinate Vectors

Consider an arbitrary, n-dimensional column vector, $\mathbf{v} \in \mathbb{R}^n$. Let $X = (\mathbf{x}_1, ..., \mathbf{x}_n)$ be a basis defining the $Ox_1...x_n$ coordinate system. Then $\mathbf{v}$ can be written in terms of the basis vectors of the X coordinate system as

$$\mathbf{v} = \sum_{i=1}^{n} v_{x_i} \hat{\mathbf{x}}_i \tag{1.8}$$

where v_{x_i} is the **component** or **coordinate** of $\mathbf{v}$ with respect to the x_i axis (i.e. in the direction defined by $\hat{\mathbf{x}}_i$). Expanding the summation,

$$\mathbf{v} = v_{x_1} \hat{\mathbf{x}}_1 + \cdots + v_{x_1} \hat{\mathbf{x}}_n$$

Using matrix algebra, we can write the equation above as

$$\mathbf{v} = \begin{bmatrix} \hat{\mathbf{x}}_1 & \cdots & \hat{\mathbf{x}}_n \end{bmatrix} \underbrace{\begin{bmatrix} v_{x_1} \\ \vdots \\ v_{x_n} \end{bmatrix}}_{[\mathbf{v}]_X} \tag{1.9}$$

The vector $[\mathbf{v}]_X \in \mathbb{R}^n$ is the **coordinate vector** of $\mathbf{v}$ **expressed** or **resolved** in basis X [26, pp. 157–162], [7].

$$\boxed{[\mathbf{v}]_X = \begin{bmatrix} v_{x_1} \\ \vdots \\ v_{x_n} \end{bmatrix}} \tag{1.10}$$

While a coordinate vector is a column vector, it is more specific than column vector. A column vector can be independent of any notion of a coordinate system, while a coordinate vector is specifically tied to a single coordinate system.

Convention 8: Coordinate vector notation.

A column vector expressed in an ordered basis should be placed in brackets and subscripted (outside the brackets) with the symbol representing the ordered basis. For example, a column vector, $\mathbf{v}$, expressed in the ordered basis X should be denoted

$$[\mathbf{v}]_X$$

If the vector has a descriptive subscript, the descriptive subscript remains inside the brackets, while the basis subscript is outside the brackets. For example, if a vector has the descriptive subscript "ex", the corresponding coordinate vector expressed in the ordered basis X is

$$[\mathbf{v}_{\text{ex}}]_X$$

As an example, consider a three-dimensional column vector, $\mathbf{v} \in \mathbb{R}^3$. Let's place $\mathbf{v}$ in the three-dimensional coordinate system $Ox_1x_2x_3$ such that its tail is located at the origin. Vector $\mathbf{v}$ has components/coordinates v_1, v_2, and v_3 directed along the three coordinate axes, defined by the basis vectors $\hat{\mathbf{x}}_1$, $\hat{\mathbf{x}}_2$, and $\hat{\mathbf{x}}_3$, respectively. For this example,

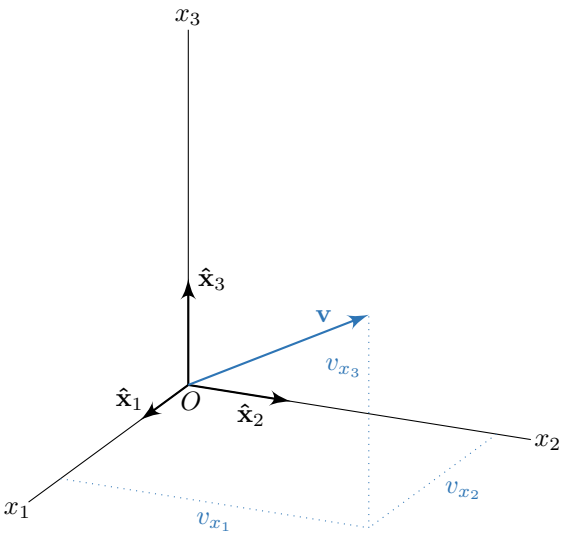


Figure 1.3: Column vector in a coordinate system.

$\mathbf{v}$ can be written in terms of the basis vectors defining $Ox_1x_2x_3$ as

$$\mathbf{v} = v_{x_1}\hat{\mathbf{x}}_1 + v_{x_2}\hat{\mathbf{x}}_2 + v_{x_3}\hat{\mathbf{x}}_3$$

and its coordinate vector expressed in X is then

$$[\mathbf{v}]_X = \begin{bmatrix} v_{x_1} \\ v_{x_2} \\ v_{x_3} \end{bmatrix}$$

1.3.1 Coordinate Basis Vectors

When we expressed a vector in a coordinate system, we obtained a coordinate vector. Similarly, when we express a basis vector in a coordinate system, we obtain a **coordinate basis vector**. This might be a bit confusing at first since basis vectors are what define coordinate systems. However, if you have multiple bases, you can express one set of

basis vectors in another basis; this will be covered more in depth in Section 1.4. As an example, consider the ordered basis

$$X = ([\hat{\mathbf{x}}_1]_Y, ..., [\hat{\mathbf{x}}_n]_Y)$$

Note that the basis vectors of X are expressed in another basis, Y.

1.3.2 The Basis Matrix

At the beginning of this section, we disregarded the basis in which basis vectors were defined when introducing the concept of a coordinate vector. Now, let's treat coordinate vectors a bit more rigorously.

Consider a coordinate system $Ox_1...x_n$ defined using the basis $X = ([\hat{\mathbf{x}}_1]_U, ..., [\hat{\mathbf{x}}_n]_U)$. Note that this time, the basis vectors of X are expressed in some other basis, $U = (\hat{\mathbf{u}}_1, ..., \hat{\mathbf{u}}_n)$. A vector expressed in U can be written as a linear combination of the basis vectors of X as

$$[\mathbf{v}]_U = \sum_{i=1}^{n} v_{x_i}[\hat{\mathbf{x}}_i]_U = v_{x_1}[\hat{\mathbf{x}}_1]_U + \cdots + v_{x_n}[\hat{\mathbf{x}}_n]_U \tag{1.11}$$

Note that $[\hat{\mathbf{x}}_i]_U$ is essentially $\hat{\mathbf{x}}_i$ projected onto $\hat{\mathbf{u}}_i$. Thus, each scalar x_i represents a coordinate with respect to the ith axis of the coordinate system defined by U.

Equivalently, we can write Eq. (1.11) using matrix algebra as

$$\mathbf{v}_U = \underbrace{\begin{pmatrix}[\hat{\mathbf{x}}_1]_U & \cdots & [\hat{\mathbf{x}}_n]_U\end{pmatrix}}_{[\mathbf{X}]_U} \underbrace{\begin{bmatrix} v_{x_1} \\ \vdots \\ v_{x_n} \end{bmatrix}}_{\mathbf{v}_X} \tag{1.12}$$

As before, we can easily recognize the coordinate vector $\mathbf{v}_X$. Now, let's define the **basis matrix** of X expressed in U as

$$\boxed{[\mathbf{X}]_U = \begin{pmatrix}[\hat{\mathbf{x}}_1]_U & \cdots & [\hat{\mathbf{x}}_n]_U\end{pmatrix}} \tag{1.13}$$

Note that since $\hat{\mathbf{x}}_i \in \mathbb{R}^n \; \forall \; i = 1, ..., n$, we have that

$$\boxed{[\mathbf{X}]_U \in \mathbb{R}^{n \times n}} \tag{1.14}$$

1.3.3 Autorepresentation

In this section, we first defined coordinate vectors in terms of "plain" basis vectors. Then, we took a step back and introduced *coordinate* basis vectors, where the basis vectors themselves were expressed in some other basis, Now, we consider the case where the basis vectors are expressed in their *own* basis (i.e. in the basis they define).

Consider the **autorepresentation** of the basis X, that is, the representation of X with respect to itself.

$$X = ([\hat{\mathbf{x}}_1]_X, ..., [\hat{\mathbf{x}}_n]_X)$$

From Eq. (1.12)

$$[\mathbf{v}]_X = \underbrace{\begin{pmatrix}[\hat{\mathbf{x}}_1]_X & \cdots & [\hat{\mathbf{x}}_n]_X\end{pmatrix}}_{[\mathbf{X}]_X} \underbrace{\begin{bmatrix} v_{x_1} \\ \vdots \\ v_{x_n} \end{bmatrix}}_{[\mathbf{v}]_X} \quad \rightarrow \quad [\mathbf{v}]_X = [\mathbf{X}]_X[\mathbf{v}]_X$$

Thus [20],

$$[\mathbf{X}]_X = \mathbf{I}_{n \times n} \quad \text{for the basis } X = ([\hat{\mathbf{x}}_1]_X, ..., [\hat{\mathbf{x}}_n]_X)$$

Convention 9: Notation for the autorepresentation of a basis.

Consider the autorepresentation of the basis X.

$$X = ([\hat{\mathbf{x}}_1]_X, ..., [\hat{\mathbf{x}}_n]_X)$$

To avoid cluttering notation, we write this ordered basis as

$$X = (\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_n)$$

In other words, if a basis is defined using coordinate basis vectors expressed in the basis itself, then we do not label the basis vectors with a basis.

$$(\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_n) \equiv ([\hat{\mathbf{x}}_1]_X, ..., [\hat{\mathbf{x}}_n]_X)$$

1.4 Change of Basis

This section is compiled from the material in [26, pp. 163–172], [49, pp. 33–34], [7], and [24].

Consider a vector $\mathbf{v} \in \mathbb{R}^n$ and two separate bases, X and Y, each with basis vectors resolved in the bases they define.

$$X = (\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_n)$$
$$Y = (\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n)$$

Supposed the tail of $\mathbf{v}$ and the tails of all the basis vectors are all located at the same point, O. The basis vectors of X and Y define the coordinate systems $Ox_1...x_n$ and $Oy_1...y_n$, respectively, both with origin O.

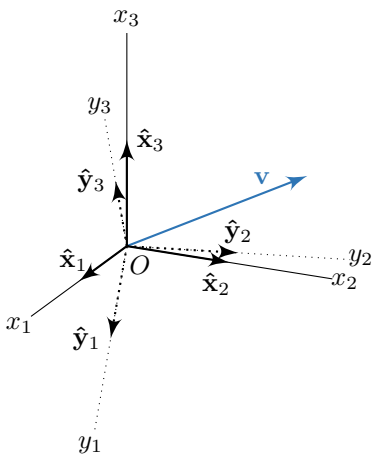


Figure 1.4: Column vector with respect to two coordinate frames.

Goals:

1. Express $\mathbf{v}$ in the $Oy_1...y_n$ coordinate system, given its coordinates in the $Ox_1...x_n$ coordinate system. This operation represents the **change of basis** from X to Y; we obtain $[\mathbf{v}]_Y$ given $[\mathbf{v}]_X$.

2. Express $\mathbf{v}$ in the $Ox_1...x_n$ coordinate system, given its coordinates in the $Oy_1...y_n$ coordinate system. This operation represents the **change of basis** from Y to X; we obtain $[\mathbf{v}]_X$ given $[\mathbf{v}]_Y$.

1.4.1 Generalized Change of Basis

Recall Eq. (1.12) defining $[\mathbf{v}]_U$ in terms of the basis matrix of X with respect to basis U, where $X = ([\hat{\mathbf{x}}_1]_U, ..., [\hat{\mathbf{x}}_n]_U)$ and $U = (\hat{\mathbf{u}}_1, ..., \hat{\mathbf{u}}_n)$.

$$[\mathbf{v}]_U = \underbrace{\begin{pmatrix}[\hat{\mathbf{x}}_1]_U & \cdots & [\hat{\mathbf{x}}_n]_U\end{pmatrix}}_{[\mathbf{X}]_U \text{ (Eq. (1.13))}} \underbrace{\begin{bmatrix} v_{x_1} \\ \vdots \\ v_{x_n} \end{bmatrix}}_{[\mathbf{v}]_X}$$

As noted in the equation above, the first matrix is just the basis matrix of X expressed in U, while the vector is the vector $\mathbf{v}$ expressed in the basis X. Thus, we have

$$[\mathbf{v}]_U = [\mathbf{X}]_U[\mathbf{v}]_X \tag{1.15}$$

Similarly, we could write $[\mathbf{v}]_U$ in terms of $[\mathbf{v}]_Y$ and $[\mathbf{Y}]_U$, where $Y = ([\hat{\mathbf{y}}_1]_U, ..., [\hat{\mathbf{y}}_n]_U)$ is another basis with its basis vectors expressed in U.

$$[\mathbf{v}]_U = [\mathbf{Y}]_U[\mathbf{v}]_Y \tag{1.16}$$

From Eqs. (1.15) and (1.16), we have

$$[\mathbf{X}]_U[\mathbf{v}]_X = [\mathbf{Y}]_U[\mathbf{v}]_Y \tag{1.17}$$

Solving for $[\mathbf{v}]_Y$ by left-multiplying both sides by $[\mathbf{Y}]_U^{-1}$,

$$[\mathbf{v}]_Y = [\mathbf{Y}]_U^{-1}[\mathbf{X}]_U[\mathbf{v}]_X$$

Below, we more formally define this change of basis.

Formal Definitions

Consider two unique bases, X and Y, which both have basis vectors expressed in a third basis U.

$$X = ([\hat{\mathbf{x}}_1]_U, ..., [\hat{\mathbf{x}}_n]_U)$$
$$Y = ([\hat{\mathbf{y}}_1]_U, ..., [\hat{\mathbf{y}}_n]_U)$$
$$U = (\hat{\mathbf{u}}_1, ..., \hat{\mathbf{u}}_n)$$

Suppose we know the basis vectors of X and Y expressed in U.

$$[\mathbf{X}]_U = \begin{pmatrix}[\hat{\mathbf{x}}_1]_U & \cdots & [\hat{\mathbf{x}}_n]_U\end{pmatrix}$$
$$[\mathbf{Y}]_U = \begin{pmatrix}[\hat{\mathbf{y}}_1]_U & \cdots & [\hat{\mathbf{y}}_n]_U\end{pmatrix}$$

The **generalized change of basis** from X to Y is defined as

$$\boxed{[\mathbf{v}]_Y = [\mathbf{Y}]_U^{-1}[\mathbf{X}]_U[\mathbf{v}]_X \qquad \text{(generalized change of basis } X \to Y\text{)}} \tag{1.18}$$

Similarly, the **generalized change of basis** from Y to X is defined as

$$\boxed{[\mathbf{v}]_X = [\mathbf{X}]_U^{-1}[\mathbf{Y}]_U[\mathbf{v}]_Y \qquad \text{(generalized change of basis } Y \to X\text{)}} \tag{1.19}$$

1.4.2 Standard Change of Basis

From Eq. (1.18), we know that the generalized change of basis from X to Y is

$$[\mathbf{v}]_Y = [\mathbf{Y}]_U^{-1}[\mathbf{X}]_U[\mathbf{v}]_X$$

where $U = (\hat{\mathbf{u}}_1, ..., \hat{\mathbf{u}}_n)$ is some third basis that both X and Y are defined with respect to. Now, let's consider the case where $U = Y$, which essentially means that we have the basis vectors of X expressed in the basis Y.

$$[\mathbf{v}]_Y = \underbrace{[\mathbf{Y}]_Y^{-1}[\mathbf{X}]_Y}_{\mathbf{I}_{n\times n}}[\mathbf{v}]_X$$

Note that $[\mathbf{Y}]_Y$ is the basis matrix of an autorepresentation, which we showed in Section 1.3.3 is just the identity matrix. Therefore, this change of basis reduces to

$$[\mathbf{v}]_Y = [\mathbf{X}]_Y[\mathbf{v}]_X$$

We can perform the same procedure with Eq. (1.19), setting $U = X$, to find

$$[\mathbf{v}]_X = [\mathbf{Y}]_X[\mathbf{v}]_Y$$

Below, we more formally define this change of basis.

Formal Definitions

Consider two unique bases, X and Y. Assume that we have the vector $[\mathbf{v}]_X$ expressed in X, together with the basis vectors of X expressed in Y. The basis vectors of X form the basis matrix

$$[\mathbf{X}]_Y = \begin{pmatrix}[\hat{\mathbf{x}}_1]_Y & \cdots & [\hat{\mathbf{x}}_n]_Y\end{pmatrix}$$

The **standard change of basis** from X to Y is defined as

$$\boxed{[\mathbf{v}]_Y = [\mathbf{X}]_Y[\mathbf{v}]_X \qquad \text{(standard change of basis } X \to Y)} \tag{1.20}$$

Similarly, assume that we have the vector $[\mathbf{v}]_Y$ expressed in Y, together with the basis vectors of Y expressed in X. The basis vectors of Y form the basis matrix

$$[\mathbf{Y}]_X = \begin{pmatrix}[\hat{\mathbf{y}}_1]_X & \cdots & [\hat{\mathbf{y}}_n]_X\end{pmatrix}$$

The **standard change of basis** from Y to X is defined as

$$\boxed{[\mathbf{v}]_X = [\mathbf{Y}]_X[\mathbf{v}]_Y \qquad \text{(standard change of basis } Y \to X)} \tag{1.21}$$

Note that we can also obtain $[\mathbf{v}]_X$ by left-multiplying both sides of Eq. (1.20) by $[\mathbf{X}]_Y^{-1}$.

$$[\mathbf{v}]_Y = [\mathbf{X}]_Y[\mathbf{v}]_X \quad \to \quad [\mathbf{X}]_Y^{-1}[\mathbf{v}]_Y = [\mathbf{X}]_Y^{-1}[\mathbf{X}]_Y[\mathbf{v}]_X \quad \to \quad [\mathbf{X}]_Y^{-1}[\mathbf{v}]_Y = [\mathbf{v}]_X$$

$$\therefore [\mathbf{v}]_X = [\mathbf{X}]_Y^{-1}[\mathbf{v}]_Y$$

Comparing this result with Eq. (1.21) implies that

$$\boxed{[\mathbf{Y}]_X = [\mathbf{X}]_Y^{-1}} \tag{1.22}$$

Repeating the same process but this time left-multiplying both sides of Eq. (1.21) by $[\mathbf{Y}]_X^{-1}$ and then comparing the result with Eq. (1.20) would result in

$$\boxed{[\mathbf{X}]_Y = [\mathbf{Y}]_X^{-1}} \tag{1.23}$$

1.5 Orthogonality

Orthogonal Vectors

Two vectors, $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^n$, are said to be orthogonal if their inner product is 0.

$$\boxed{\mathbf{v}_1^T \mathbf{v}_2 = 0 \quad \rightarrow \quad \mathbf{v}_1 \in \mathbb{R}^n \text{ and } \mathbf{v}_2 \in \mathbb{R}^n \text{ are orthogonal}} \tag{1.24}$$

Orthonormal Sets

An **orthonormal set** is a set of unit vectors that are all orthogonal to one another. Let $U = \{\hat{\mathbf{u}}_1, ..., \hat{\mathbf{u}}_n\}$ be a set of n vectors such that

$$\hat{\mathbf{u}}_i^T \hat{\mathbf{u}}_j = \delta_{ij}$$

where

$$\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Then U is an orthonormal set.

Note that the condition above implies that all the vectors are both orthogonal to one another and have magnitude 1. For the case where $i \neq j$, $\hat{\mathbf{u}}_i^T \hat{\mathbf{u}}_j = 0$ implies that the vectors are orthogonal. For the case where $i = j$, $\hat{\mathbf{u}}_i^T \hat{\mathbf{u}}_i = 1$ implies that $\hat{\mathbf{u}}_i$ is a unit vector, since $\hat{\mathbf{u}}_i^T \hat{\mathbf{u}}_i = \|\hat{\mathbf{u}}_i\|^2$.

Orthogonal Matrices

A square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is said to be **orthogonal** if its column vectors form an orthonormal set. Let $\mathbf{a}_i$ be the ith column vector of $\mathbf{A}$.

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_i & \cdots & \mathbf{a}_n \end{bmatrix}$$

$\mathbf{A}$ is an orthogonal matrix if

$$\|\mathbf{a}_i\| = 1 \; \forall \; i = 1, .., n$$
$$\mathbf{a}_i^T \mathbf{a}_j = 0 \; \forall \; i, j = 1, .., n, \; i \neq j$$

Properties of Orthogonal Matrices

Consider multiplying $\mathbf{A}$ by its transpose from the left. Writing the matrices in terms of their column vectors,

$$\mathbf{A}^T \mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix}^T \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_n^T \end{bmatrix} \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^T \mathbf{a}_1 & \mathbf{a}_1^T \mathbf{a}_2 & \cdots & \mathbf{a}_1^T \mathbf{a}_n \\ \mathbf{a}_2^T \mathbf{a}_1 & \mathbf{a}_2^T \mathbf{a}_2 & \cdots & \mathbf{a}_2^T \mathbf{a}_n \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_n^T \mathbf{a}_1 & \mathbf{a}_n^T \mathbf{a}_2 & \cdots & \mathbf{a}_n^T \mathbf{a}_n \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

It follows that an orthogonal matrix has the property

$$\boxed{\mathbf{A}^T \mathbf{A} = \mathbf{A} \mathbf{A}^T = \mathbf{I}_{n \times n}} \tag{1.25}$$

where $\mathbf{I}_{n\times n}$ is the $n \times n$ identity matrix.

We also know that $\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}_{n\times n}$, which when compared to Eq. (1.25) yields the property

$$\boxed{\mathbf{A}^{-1} = \mathbf{A}^T} \tag{1.26}$$

Eq. (1.26) is an extremely important property because it is computationally much more efficient to calculate the transpose of a matrix than its inverse.

Next, consider the determinant of $\mathbf{A}\mathbf{A}^T$. We know that

$$\det\left(\mathbf{I}_{n\times n}\right) = 1$$

By the definition of the determinant, we know that

$$\det\left(\mathbf{A}\mathbf{A}^T\right) = \det\left(\mathbf{A}\right)\det\left(\mathbf{A}^T\right)$$

Since $\det\left(\mathbf{A}^T\right) = \det\left(\mathbf{A}\right)$ for any square matrix $\mathbf{A}$,

$$\det\left(\mathbf{A}\mathbf{A}^T\right) = \det\left(\mathbf{A}\right)^2$$

Since $\mathbf{A}\mathbf{A}^T = \mathbf{I}_{n\times n}$,

$$\det\left(\mathbf{I}_{n\times n}\right) = \det\left(\mathbf{A}\right)^2 \quad \rightarrow \quad \det\left(\mathbf{A}\right)^2 = 1$$

$$\boxed{\left|\det\left(\mathbf{A}\right)\right| = 1} \tag{1.27}$$

In Eq. (1.27), $|\cdot|$ denotes an absolute value while $\det(\cdot)$ denotes a determinant; this is essential to note because $|\cdot|$ is often used to denote the determinant [43].

> If a matrix is orthogonal, its determinant is either -1 or 1. ***However***, if the determinant of a square matrix is -1 or 1, it does not imply that the matrix is orthogonal [14].

Now, consider two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and the orthogonal matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$. Taking the inner product of $\mathbf{A}\mathbf{x}$ with $\mathbf{A}\mathbf{y}$,

$$\left(\mathbf{A}\mathbf{x}\right)^T\left(\mathbf{A}\mathbf{y}\right) = \mathbf{x}^T \underbrace{\mathbf{A}^T\mathbf{A}}_{\mathbf{I}_{n\times n}}\mathbf{y}$$

$$\boxed{\left(\mathbf{A}\mathbf{x}\right)^T\left(\mathbf{A}\mathbf{y}\right) = \mathbf{x}^T\mathbf{y}} \tag{1.28}$$

In the case that $\mathbf{x} = \mathbf{y}$,

$$\left(\mathbf{A}\mathbf{x}\right)^T\mathbf{A}\mathbf{x} = \mathbf{x}^T\mathbf{x} = \|\mathbf{x}\|_2^2$$

We also know that

$$\left(\mathbf{A}\mathbf{x}\right)^T\left(\mathbf{A}\mathbf{x}\right) = \|\mathbf{A}\mathbf{x}\|^2$$

Thus, we have [26, pp. 263–268][52, p. 15]

$$\boxed{\|\mathbf{A}\mathbf{x}\| = \|\mathbf{x}\|} \tag{1.29}$$

Product of Orthogonal Matrices

Consider two orthogonal matrices $\mathbf{A}$ and $\mathbf{B}$. Their product is given by $\mathbf{A}\mathbf{B}$. Thus, from Eq. (1.26), we know that if $\left(\mathbf{A}\mathbf{B}\right)^T\left(\mathbf{A}\mathbf{B}\right) = \mathbf{I}$, then the matrix product $\mathbf{A}\mathbf{B}$ is also orthogonal.

$$\left(\mathbf{A}\mathbf{B}\right)^T\left(\mathbf{A}\mathbf{B}\right) = \left(\mathbf{B}^T\mathbf{A}^T\right)\left(\mathbf{A}\mathbf{B}\right) = \mathbf{B}^T\left(\mathbf{A}^T\mathbf{A}\right)\mathbf{B} = \mathbf{B}^T\mathbf{I}\mathbf{B} = \mathbf{B}^T\mathbf{B} = \mathbf{I}$$

Thus, the product of two orthogonal matrices is also an orthogonal matrix [57].

1.5.1 Orthonormal Bases and Cartesian Coordinate Systems

Consider a ordered basis $X_U = ([\hat{\mathbf{x}}_1]_U, ..., [\hat{\mathbf{x}}_n]_U)$ with basis vectors expressed in another basis U with a corresponding basis matrix

$$[\mathbf{X}]_U = \begin{pmatrix} [\hat{\mathbf{x}}_1]_U & \cdots & [\hat{\mathbf{x}}_n]_U \end{pmatrix}$$

If $[\mathbf{X}]_U$ is an orthogonal matrix, or equivalently, if all the basis vectors defining X are mutually orthogonal, then X is an ordered basis for a **Cartesian coordinate system**.

Let's consider the case where X and Y are both orthonormal bases defining coordinate systems, with corresponding coordinate basis matrices

$$[\mathbf{X}]_U = \begin{pmatrix} [\hat{\mathbf{x}}_1]_U & \cdots & [\hat{\mathbf{x}}_n]_U \end{pmatrix}, \qquad [\mathbf{Y}]_U = \begin{pmatrix} [\hat{\mathbf{y}}_1]_U & \cdots & [\hat{\mathbf{y}}_n]_U \end{pmatrix}$$

Since the basis vectors of X and Y both form orthonormal sets, $[\mathbf{X}]_U$ and $[\mathbf{Y}]_U$ are orthogonal matrices, implying (from Eq. (1.26)) that $[\mathbf{X}]_U^{-1} = [\mathbf{X}]_U^T$ and $[\mathbf{Y}]_U^{-1} = [\mathbf{Y}]_U^T$. The generalized change of basis formulas from Section 1.4.1 can then be simplified to

$$\boxed{[\mathbf{v}]_Y = [\mathbf{Y}]_U^T [\mathbf{X}]_U [\mathbf{v}]_X \qquad \text{(generalized change of basis } X \to Y \text{ where } Y \text{ is an orthonormal basis)}} \tag{1.30}$$

$$\boxed{[\mathbf{v}]_X = [\mathbf{X}]_U^T [\mathbf{Y}]_U [\mathbf{v}]_Y \qquad \text{(generalized change of basis } Y \to X \text{ where } X \text{ is an orthonormal basis)}} \tag{1.31}$$

Similarly, we can replace the inverses in Eqs. (1.22) and (1.23) to obtain

$$\boxed{[\mathbf{Y}]_X = [\mathbf{X}]_Y^T \quad \text{(if } X \text{ is an orthonormal basis)}} \tag{1.32}$$

$$\boxed{[\mathbf{X}]_Y = [\mathbf{Y}]_X^T \quad \text{(if } Y \text{ is an orthonormal basis)}} \tag{1.33}$$

Note that these definitions are much faster computationally since transposing a matrix requires much fewer operations than performing a matrix inversion.

1.6 Rotation Matrices

Recall from Section 1.5.1 that a Cartesian coordinate system is defined by an orthonormal basis. While we already simplified the generalized change of basis for Cartesian coordinate systems, there are special ways we can perform a change of basis in a three-dimensional Euclidian space.

In a three-dimensional Euclidian space, the change of basis from one Cartesian coordinate system to another can be represented using rotations.

1.6.1 Passive vs. Active Rotations

When dealing with rotations of coordinate spaces and vectors, there are two common ways that rotations are performed:

1. **Passive Rotation:** The original coordinate system is rotated by some angle θ about one of its axes, while any vector or matrix quantities remain constant, i.e. *passive*, with respect to the original coordinate systems. The vector and matrix quantities are then expressed in the new coordinate system.
2. **Active Rotation:** The coordinate system remains stationary and the vector or matrix *actively* rotates with respect to the original coordinate system.

In fields such as computer graphics and video games, active rotations are generally used because there is usually some object moving in a fixed coordinate system. However, in the field of dynamics, specifically aerospace dynamics, we typically use passive rotations, since we will have a vector that we want to *resolve* or *express* in different coordinate systems [1].

1.6.2 Elementary Rotations

Consider the $Ox_1x_2x_3$ coordinate system. We refer to the three axes of the this coordinate system as either the **first**, **second**, or **third axes**.

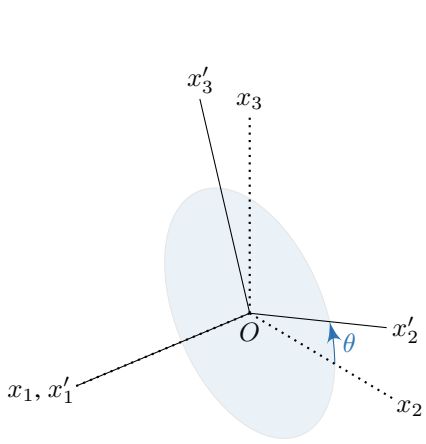
Convention 10: Numbering axes.

1. $x_1 =$ 1st axis
2. $x_2 =$ 2nd axis
3. $x_3 =$ 3rd axis

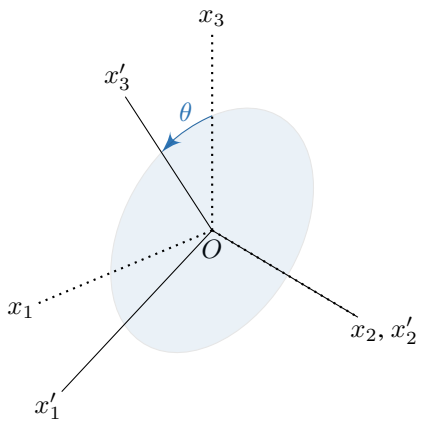
We define the **rotation matrix** for a counterclockwise rotation of θ about the ith axis of a coordinate system as $\mathbf{R}_i(\theta)$. There are three **elementary rotations**, each about a different axes, encoded by $\mathbf{R}_1$, $\mathbf{R}_2$, and $\mathbf{R}_3$, respectively:

1. $\mathbf{R}_1(\theta)$: Encodes counterclockwise rotation by θ of the 2nd and 3rd axes (x_2x_3-plane) about the 1st axis (x_1).
2. $\mathbf{R}_2(\theta)$: Encodes counterclockwise rotation by θ of the 1st and 3rd axes (x_1x_3-plane) about the 2nd axis (x_2).
3. $\mathbf{R}_3(\theta)$: Encodes counterclockwise rotation by θ of the 1st and 2nd axes (x_1x_2-plane) about the 3rd axis (x_3).

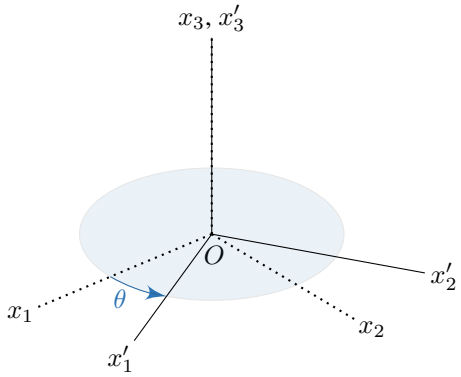
These rotations are illustrated in Fig. 1.5. The rotation matrices $\mathbf{R}_1(\theta)$, $\mathbf{R}_2(\theta)$, and $\mathbf{R}_3(\theta)$ are defined by Eqs. (1.34),



(a) Rotation about 1st axis.



(b) Rotation about 2nd axis.



(c) Rotation about 3rd axis.

Figure 1.5: Elementary rotations.

(1.35), and (1.36), respectively [55, p. 162], [17], [47].

$$\boxed{\mathbf{R}_1(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}} \tag{1.34}$$

$$\boxed{\mathbf{R}_2(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}} \tag{1.35}$$

$$\boxed{\mathbf{R}_3(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}} \tag{1.36}$$

We formalize these simple equations as algorithms since they will be used often in other algorithms.

Algorithm 1: `rot1`
Rotation matrix for a passive rotation about the 1st axis.

Inputs:

- $\theta \in \mathbb{R}$ - angle of rotation $[\text{rad}]$

Procedure:

1. Precompute the trigonometric functions.

$$c = \cos\theta$$
$$s = \sin\theta$$

2. Construct the rotation matrix.

$$\mathbf{R}_1(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix}$$

Outputs:

- $\mathbf{R}_1(\theta) \in \mathbb{R}^{3\times 3}$ - rotation matrix about 1st axis (passive)

Test Cases:

- See Appendix A.1.1.

Algorithm 2: `rot2`
Rotation matrix for a passive rotation about the 2nd axis.

Inputs:

- $\theta \in \mathbb{R}$ - angle of rotation $[\text{rad}]$

Procedure:

1. Precompute the trigonometric functions.

$$c = \cos\theta$$
$$s = \sin\theta$$

2. Construct the rotation matrix.

$$\mathbf{R}_2(\theta) = \begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix}$$

Outputs:

- $\mathbf{R}_2(\theta) \in \mathbb{R}^{3\times 3}$ - rotation matrix about 2nd axis (passive)

Test Cases:

- See Appendix A.1.1.

Algorithm 3: `rot3`
Rotation matrix for a passive rotation about the 3rd axis.

Inputs:

- $\theta \in \mathbb{R}$ - angle of rotation $[\text{rad}]$

Procedure:

1. Precompute the trigonometric functions.

$$c = \cos\theta$$
$$s = \sin\theta$$

2. Construct the rotation matrix.

$$\mathbf{R}_3(\theta) = \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Outputs:

- $\mathbf{R}_3(\theta) \in \mathbb{R}^{3\times 3}$ - rotation matrix about 3rd axis (passive)

Test Cases:

- See Appendix A.1.1.

1.6.3 Properties of Elementary Rotation Matrices

Let $X = (\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \hat{\mathbf{x}}_3)$ and $X' = (\hat{\mathbf{x}}'_1, \hat{\mathbf{x}}'_2, \hat{\mathbf{x}}_3)$ be two ordered bases that are both orthonormal bases, where X' is defined by rotating X about the x_3 axis by an angle θ. Consider a vector, $[\mathbf{v}]_X$, resolved in the coordinate system defined by X. To resolve that same vector in the coordinate system defined by X', we could perform the change of basis as defined by Eq. (1.20).

$$[\mathbf{v}]_{X'} = \mathbf{X}_{X'}[\mathbf{v}]_X$$

However, we can note that this same change of basis can be performed by the elementary rotation matrix $\mathbf{R}_3(\theta)$.

$$[\mathbf{v}]_{X'} = \mathbf{R}_3(\theta)[\mathbf{v}]_X$$

Since $\mathbf{X}_{X'}$ is orthogonal (X is an orthonormal basis), and since $\mathbf{R}_3(\theta) = \mathbf{X}_{X'}$, we know that $\mathbf{R}_3(\theta)$ is orthogonal. In general,

Elementary rotation matrices, $\mathbf{R}_i(\theta)$, are orthogonal matrices.

Consider defining an elementary rotation matrix using a negative angle.

$$\mathbf{R}_1(-\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(-\theta) & \sin(-\theta) \\ 0 & -\sin(-\theta) & \cos(-\theta) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}^T = \mathbf{R}_1(\theta)^T$$

$$\mathbf{R}_2(-\theta) = \begin{bmatrix} \cos(-\theta) & 0 & -\sin(-\theta) \\ 0 & 1 & 0 \\ \sin(-\theta) & 0 & \cos(-\theta) \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}^T = \mathbf{R}_2(\theta)^T$$

$$\mathbf{R}_3(-\theta) = \begin{bmatrix} \cos(-\theta) & \sin(-\theta) & 0 \\ -\sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}^T = \mathbf{R}_3(\theta)^T$$

Thus, in general, we have

$$\mathbf{R}_i(-\theta) = \mathbf{R}_i(\theta)^T$$

However, we also know that since $\mathbf{R}_i(\theta)$ is orthogonal, its inverse is equal to its transpose. Therefore, we have

$$\boxed{\mathbf{R}_i(\theta)^{-1} = \mathbf{R}_i(\theta)^T = \mathbf{R}_i(-\theta)} \tag{1.37}$$

Since elementary rotation matrices are orthogonal, they also share all the other properties of orthogonal matrices as summarized in Section 1.5. Note that in the properties below, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^3$.

$$\boxed{|\det[\mathbf{R}_i(\theta)]| = 1} \tag{1.38}$$

$$\boxed{\mathbf{R}_i(\theta)^T\mathbf{R}_i(\theta) = \mathbf{R}_i(\theta)\mathbf{R}_i(\theta)^T = \mathbf{I}_{3\times 3}} \tag{1.39}$$

$$\boxed{[\mathbf{R}_i(\theta)\mathbf{x}]^T[\mathbf{R}_i(\theta)\mathbf{y}] = \mathbf{x}^T\mathbf{y}} \tag{1.40}$$

$$\boxed{\|\mathbf{R}_i(\theta)\mathbf{x}\| = \|\mathbf{x}\|} \tag{1.41}$$

Since $\mathbf{R}_i(-\theta) = \mathbf{R}_i(\theta)^T$, we can also get an alternate version of the property presented by Eq. (1.39).

$$\boxed{\mathbf{R}_i(\theta)\mathbf{R}_i(-\theta) = \mathbf{I}_{3\times 3}} \tag{1.42}$$

1.6.4 Sequential Rotations

Consider an arbitrary vector $\mathbf{v} \in \mathbb{R}^3$ resolved with respect to the ordered basis $S = (\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$, where S defines the coordinate system $Oxyz$.

$$\mathbf{v}_S = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

First, rotating S by θ_1 about the 1st axis (x), we obtain a new coordinate system $Ox'y'z'$ defined by the ordered basis $S' = (\hat{\mathbf{x}}', \hat{\mathbf{y}}', \hat{\mathbf{z}}')$. Expressing $\mathbf{v}$ in the $Ox'y'z'$ coordinate system (i.e. with respect to the ordered basis S'),

$$\mathbf{v}_{S'} = \mathbf{R}_1(\theta_1)\mathbf{v}_S$$

Next, rotating S' by θ_2 about the 2nd axis (y'), we obtain a third coordinate system, $Ox''y''z''$ defined by the ordered basis $S'' = (\hat{\mathbf{x}}'', \hat{\mathbf{y}}'', \hat{\mathbf{z}}'')$. Expressing $\mathbf{v}$ in the $Ox''y''z''$ coordinate system (i.e. with respect to the ordered basis S''),

$$\begin{aligned}\mathbf{v}_{S''} &= \mathbf{R}_2(\theta_2)\mathbf{v}_{S'} \\ &= \mathbf{R}_2(\theta_2)\mathbf{R}_1(\theta_1)\mathbf{v}_S\end{aligned}$$

In general, for a rotation sequence $i \to j \to k$ (where i, j, and k can be either 1, 2, or 3, i.e. representing an axis), we define

$$\boxed{\mathbf{R}_{ijk}(\theta_1, \theta_2, \theta_3) = \mathbf{R}_k(\theta_3)\mathbf{R}_j(\theta_2)\mathbf{R}_i(\theta_1) \qquad \text{(for the rotation sequence } i \to j \to k\text{)}} \tag{1.43}$$

Convention 11: Sequential rotation.

The ijk rotation sequence ($i \to j \to k$) is described by the rotation matrix

$$\mathbf{R}_{ijk}(\theta_1, \theta_2, \theta_3)$$

where

1. The angles are input in the order the rotations are applied, so their subscript corresponds to the current rotation. For example, θ_2 is used for the second rotation.
2. All angles assumes counterclockwise is positive.
3. The rotations are applied in the order of the subscript:
   (a) First rotation is about the ith axis.
   (b) Second rotation is about the jth axis.
   (c) Third rotation is about the kth axis.

There are two specific rotation sequences that are typically used in aerospace simulation. The two sequences are described in detail below.

3-2-1 Rotation sequence

The 3-2-1 ($3 \to 2 \to 1$) Rotation sequence consists of the following steps:

1. rotation of θ_1 about z (3rd axis)
2. rotation of θ_2 about y' (2nd axis)
3. rotation of θ_3 about x'' (1st axis)

The rotation matrix for the 3-2-1 sequence is [17][2], [56, pp. 763-764][3]

$$\begin{aligned}\mathbf{R}_{321}(\theta_1, \theta_2, \theta_3) &= \mathbf{R}_1(\theta_3)\mathbf{R}_2(\theta_2)\mathbf{R}_3(\theta_1) \\ &= \begin{bmatrix}\cos\theta_3 & \sin\theta_3 & 0 \\ -\sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 1 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2\end{bmatrix}\begin{bmatrix}1 & 0 & 0 \\ 0 & \cos\theta_1 & \sin\theta_1 \\ 0 & -\sin\theta_1 & \cos\theta_1\end{bmatrix}\end{aligned}$$

$$\boxed{\mathbf{R}_{321}(\theta_1, \theta_2, \theta_3) = \begin{bmatrix}\cos\theta_2\cos\theta_1 & \cos\theta_2\sin\theta_1 & -\sin\theta_2 \\ -\cos\theta_3\sin\theta_1 + \sin\theta_3\sin\theta_2\cos\theta_1 & \cos\theta_3\cos\theta_1 + \sin\theta_3\sin\theta_2\sin\theta_1 & \sin\theta_3\cos\theta_2 \\ \sin\theta_3\sin\theta_1 + \cos\theta_3\sin\theta_2\cos\theta_1 & -\sin\theta_3\cos\theta_1 + \cos\theta_3\sin\theta_2\sin\theta_1 & \cos\theta_3\cos\theta_2\end{bmatrix}} \tag{1.44}$$

We formalize Eq. (1.44) as Algorithm 4 below. Note that in its implementation, we precompute the trigonometric functions *before* populating the matrix to decrease the computational cost.

[2] In this reference, $\theta_1 = \psi$, $\theta_2 = \theta$, and $\theta_3 = \phi$.
[3] In this reference, $\theta_1 = \phi$, $\theta_2 = \theta$, and $\theta_3 = \psi$.

Algorithm 4: `rot321`
Rotation matrix for the 3-2-1 rotation sequence.

Inputs:
- $\theta_1 \in \mathbb{R}$ - angle for first rotation (about 3rd axis) [rad]
- $\theta_2 \in \mathbb{R}$ - angle for second rotation (about 2nd axis) [rad]
- $\theta_3 \in \mathbb{R}$ - angle for third rotation (about 1st axis) [rad]

Procedure:
1. Precompute the trigonometric functions.

$$
\begin{aligned}
s_1 &= \sin\theta_1 \\
c_1 &= \cos\theta_1 \\
s_2 &= \sin\theta_2 \\
c_2 &= \cos\theta_2 \\
s_3 &= \sin\theta_3 \\
c_3 &= \cos\theta_3
\end{aligned}
$$

2. Construct the rotation matrix.

$$
\mathbf{R}_{321}(\theta_1,\theta_2,\theta_3) = \begin{bmatrix} c_2c_1 & c_2s_1 & -s_2 \\ -c_3s_1 + s_3s_2c_1 & c_3c_1 + s_3s_2s_1 & s_3c_2 \\ s_3s_1 + c_3s_2c_1 & -s_3c_1 + c_3s_2s_1 & c_3c_2 \end{bmatrix}
$$

Outputs:
- $\mathbf{R}_{321}(\theta_1,\theta_2,\theta_3) \in \mathbb{R}^{3\times3}$ - rotation matrix for 3-2-1 rotation sequence

Test Cases:
- See Appendix A.1.1.

3-1-3 Rotation sequence

The 3-1-3 ($3 \to 1 \to 3$) Rotation sequence consists of the following steps:
1. rotation of θ_1 about x_3 (3rd axis)
2. rotation of θ_2 about x_1' (1st axis)
3. rotation of θ_3 about x_3'' (3rd axis)

The rotation matrix for the 3-1-3 sequence is [17], [56, pp. 763-764][4]

$$
\begin{aligned}
\mathbf{R}_{313}(\theta_1,\theta_2,\theta_3) &= \mathbf{R}_3(\theta_3)\mathbf{R}_1(\theta_2)\mathbf{R}_3(\theta_1) \\
&= \begin{bmatrix} \cos\theta_3 & \sin\theta_3 & 0 \\ -\sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_2 & \sin\theta_2 \\ 0 & -\sin\theta_2 & \cos\theta_2 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

[4] In this reference, $\theta_1 = \phi$, $\theta_2 = \theta$, and $\theta_3 = \psi$.

$$\mathbf{R}_{313}(\theta_1,\theta_2,\theta_3) = \begin{bmatrix} \cos\theta_3\cos\theta_1 - \sin\theta_3\cos\theta_2\sin\theta_1 & \cos\theta_3\sin\theta_1 + \sin\theta_3\cos\theta_2\cos\theta_1 & \sin\theta_3\sin\theta_2 \\ -\sin\theta_3\cos\theta_1 - \cos\theta_3\cos\theta_2\sin\theta_1 & -\sin\theta_3\sin\theta_1 + \cos\theta_3\cos\theta_2\cos\theta_1 & \cos\theta_3\sin\theta_2 \\ \sin\theta_2\sin\theta_1 & -\sin\theta_2\cos\theta_1 & \cos\theta_2 \end{bmatrix} \tag{1.45}$$

We formalize Eq. (1.45) as Algorithm 5 below. Note that in its implementation, we precompute the trigonometric functions *before* populating the matrix to decrease the computational cost.

Algorithm 5: `rot313`
Rotation matrix for the 3-1-3 rotation sequence.

Inputs:
- $\theta_1 \in \mathbb{R}$ - angle for first rotation (about 3rd axis) [rad]
- $\theta_2 \in \mathbb{R}$ - angle for second rotation (about 1st axis) [rad]
- $\theta_3 \in \mathbb{R}$ - angle for third rotation (about 3rd axis) [rad]

Procedure:
1. Precompute the trigonometric functions.

$$\begin{aligned} s_1 &= \sin\theta_1 \\ c_1 &= \cos\theta_1 \\ s_2 &= \sin\theta_2 \\ c_2 &= \cos\theta_2 \\ s_3 &= \sin\theta_3 \\ c_3 &= \cos\theta_3 \end{aligned}$$

2. Construct the rotation matrix.

$$\mathbf{R}_{313}(\theta_1,\theta_2,\theta_3) = \begin{bmatrix} c_3c_1 - s_3c_2s_1 & c_3s_1 + s_3c_2c_1 & s_3s_2 \\ -s_3c_1 - c_3c_2s_1 & -s_3s_1 + c_3c_2c_1 & c_3s_2 \\ s_2s_1 & -s_2c_1 & c_2 \end{bmatrix}$$

Outputs:
- $\mathbf{R}_{313}(\theta_1,\theta_2,\theta_3) \in \mathbb{R}^{3\times3}$ - rotation matrix for 3-1-3 rotation sequence

1.6.5 Properties of Sequential Rotation Matrices

A sequential rotation matrix is defined as

$$\mathbf{R}_{ijk}(\theta_1,\theta_2,\theta_3) = \mathbf{R}_k(\theta_3)\mathbf{R}_j(\theta_2)\mathbf{R}_i(\theta_1)$$

$\mathbf{R}_i(\theta_1)$, $\mathbf{R}_j(\theta_2)$, and $\mathbf{R}_k(\theta_3)$ are all elementary rotation matrices, and in Section 1.6.3 we showed that elementary rotation matrices are orthogonal matrices. Additionally, from Section 1.5, we know that taking the product of orthogonal matrices results in an orthogonal matrix. Thus, we know that

Sequential rotation matrices, $\mathbf{R}_{ijk}(\theta_1,\theta_2,\theta_3)$, are orthogonal matrices.

Consider taking the transpose of $\mathbf{R}_{ijk}(\theta_1,\theta_2,\theta_3)$.

$$\begin{aligned} \mathbf{R}_{ijk}(\theta_1,\theta_2,\theta_3)^T &= [\mathbf{R}_k(\theta_3)\mathbf{R}_j(\theta_2)\mathbf{R}_i(\theta_1)]^T = \mathbf{R}_i(\theta_1)^T\mathbf{R}_j(\theta_2)^T\mathbf{R}_k(\theta_3)^T = \mathbf{R}_i(-\theta_1)\mathbf{R}_j(-\theta_2)\mathbf{R}_k(-\theta_3) \\ &= \mathbf{R}_{kji}(-\theta_3,-\theta_2,-\theta_1) \end{aligned}$$

Since sequential rotation matrices are orthogonal matrices, their transpose is also equal to their inverse. Thus,

$$\boxed{\mathbf{R}_{ijk}(\theta_1,\theta_2,\theta_3)^{-1} = \mathbf{R}_{ijk}(\theta_1,\theta_2,\theta_3)^T = \mathbf{R}_{kji}(-\theta_3,-\theta_2,-\theta_1) = \mathbf{R}_i(-\theta_1)\mathbf{R}_j(-\theta_2)\mathbf{R}_k(-\theta_3)} \tag{1.46}$$

1.7 Skew Symmetry

In Section 1.5, we covered orthogonal matrices in-depth. Another type of matrix that arises often in dynamics is the **skew-symmetrix matrix**. A skew-symmetric matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ satisfies the property

$$\boxed{\mathbf{A}^T = -\mathbf{A}} \tag{1.47}$$

which also implies that

$$\boxed{\mathbf{A} + \mathbf{A}^T = \mathbf{0}_{n\times n}} \tag{1.48}$$

In the context of aerospace dynamics, the special case where $\mathbf{A} \in \mathbb{R}^{3\times 3}$ is the one of greatest importance. In the 3×3 case, all skew-symmetric matrices are of the form [45]

$$\boxed{\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 0 & -a_{21} & a_{13} \\ a_{21} & 0 & -a_{32} \\ -a_{13} & a_{32} & 0 \end{bmatrix}} \tag{1.49}$$

We can verify that $\mathbf{A} + \mathbf{A}^T = \mathbf{0}_{3\times 3}$.

$$\begin{aligned} \mathbf{A} + \mathbf{A}^T &= \begin{bmatrix} 0 & -a_{21} & a_{13} \\ a_{21} & 0 & -a_{32} \\ -a_{13} & a_{32} & 0 \end{bmatrix} + \begin{bmatrix} 0 & -a_{21} & a_{13} \\ a_{21} & 0 & -a_{32} \\ -a_{13} & a_{32} & 0 \end{bmatrix}^T \\ &= \begin{bmatrix} 0 & -a_{21} & a_{13} \\ a_{21} & 0 & -a_{32} \\ -a_{13} & a_{32} & 0 \end{bmatrix} + \begin{bmatrix} 0 & a_{21} & -a_{13} \\ -a_{21} & 0 & a_{32} \\ a_{13} & -a_{32} & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ &= \mathbf{0}_{3\times 3} \end{aligned}$$

1.8 Vector Operations and Properties in Cartesian Coordinate Systems

Let $Oxyz$ be a Cartesian coordinate system defined by the orthonormal ordered basis $A = (\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$. Now, consider two vectors (illustated in Fig. 1.6), $\mathbf{a}$ and $\mathbf{b}$, which can be written as linear combinations of the basis vectors of E.

$$\begin{aligned} \mathbf{a} &= a_x\hat{\mathbf{x}} + a_y\hat{\mathbf{y}} + a_z\hat{\mathbf{z}} \\ \mathbf{b} &= b_x\hat{\mathbf{x}} + b_y\hat{\mathbf{y}} + b_z\hat{\mathbf{z}} \end{aligned}$$

Below, we compile many properties of vectors in Cartesian coordinate systems [50]

> $\mathbf{a}$ and $\mathbf{b}$ are technically coordinate vectors since they are expressed in a coordinate system. However, since they are both expressed in the *same* coordinate system, it is

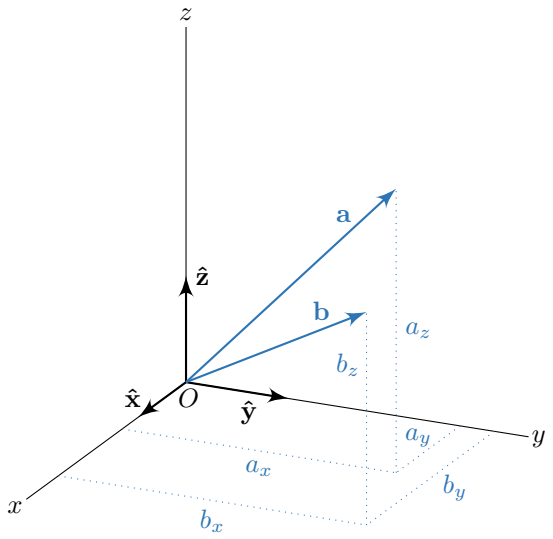


Figure 1.6: Vectors in a Cartesian coordinate system.

not important to include the subscript labeling the frame the coordinate vectors are expressed in.

Vector addition

To add two vectors, we can simply add the individual components.

$$\mathbf{a} + \mathbf{b} = \begin{bmatrix} a_x + b_x \\ a_y + b_y \\ a_z + b_z \end{bmatrix} \tag{1.50}$$

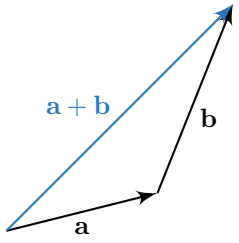


Figure 1.7: Vector addition.

Vector subtraction

To find the difference between two vectors, we can simply find the differences between the individual components.

$$\mathbf{a} - \mathbf{b} = \begin{bmatrix} a_x - b_x \\ a_y - b_y \\ a_z - b_z \end{bmatrix} \tag{1.51}$$

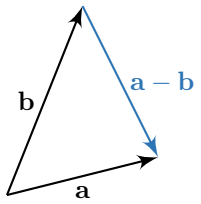


Figure 1.8: Vector subtraction.

Dot (scalar) product

The **dot (scalar) product** can be written either in terms of the components of $\mathbf{a}$ and $\mathbf{b}$, or in terms of their magnitudes and the angle between them.

$$\boxed{\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T\mathbf{b} = a_x b_x + a_y b_y + a_z b_z = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta} \tag{1.52}$$

The dot product also exhibits the following properties (where $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ are vectors and k is an arbitrary constant).

$$\boxed{\begin{aligned} \mathbf{a} \cdot \mathbf{a} &= \|\mathbf{a}\|^2 && \text{(1.53a)} \\ \mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) &= \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c} && \text{(1.53b)} \\ \mathbf{0} \cdot \mathbf{a} &= 0 && \text{(1.53c)} \\ \mathbf{a} \cdot \mathbf{b} &= \mathbf{b} \cdot \mathbf{a} && \text{(1.53d)} \\ (k\mathbf{a}) \cdot \mathbf{b} &= k(\mathbf{a} \cdot \mathbf{b}) = \mathbf{a} \cdot (k\mathbf{b}) && \text{(1.53e)} \end{aligned}}$$

Finally, we can also find the following identities for the dot products between the unit vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$.

$$\boxed{\begin{aligned} \hat{\mathbf{x}} \cdot \hat{\mathbf{y}} = \hat{\mathbf{y}} \cdot \hat{\mathbf{z}} = \hat{\mathbf{z}} \cdot \hat{\mathbf{x}} &= 0 && \text{(1.54a)} \\ \hat{\mathbf{x}} \cdot \hat{\mathbf{x}} = \hat{\mathbf{y}} \cdot \hat{\mathbf{y}} = \hat{\mathbf{z}} \cdot \hat{\mathbf{z}} &= 1 && \text{(1.54b)} \end{aligned}}$$

Vector magnitude

The magnitude of a vector $\mathbf{a}$ can be denoted as either a or $\|\mathbf{a}\|$ and is found by taking the 2-norm of $\mathbf{a}$. Note that instead of using $\|\cdot\|_2$ to denote the 2-norm, we just use $\|\cdot\|$. This is customary in the study of vectors in $\mathbb{R}^3$.

$$\boxed{a = \|\mathbf{a}\| = \sqrt{\mathbf{a} \cdot \mathbf{a}} = \sqrt{\mathbf{a}^T\mathbf{a}} = \sqrt{a_x^2 + a_y^2 + a_z^2}} \tag{1.55}$$

Unit vector in direction of $\mathbf{a}$

The **unit vector** $\hat{\mathbf{a}}$ corresponding to the vector $\mathbf{a}$ is a vector of magnitude 1 in the direction of $\mathbf{a}$.

$$\boxed{\hat{\mathbf{a}} = \frac{\mathbf{a}}{\|\mathbf{a}\|} = \frac{\mathbf{a}}{a} = \frac{\mathbf{a}}{\sqrt{a_x^2 + a_y^2 + a_z^2}}} \tag{1.56}$$

Cross (vector) product

The **cross (vector) product** of $\mathbf{a}$ and $\mathbf{b}$ is a vector multiplication that produces a third vector that is normal/orthogonal/perpendicular to both $\mathbf{a}$ and $\mathbf{b}$.

$$\boxed{\mathbf{a} \times \mathbf{b} = \begin{bmatrix} b_z a_y - a_z b_y \\ b_x a_z - b_z a_x \\ b_y a_x - b_x a_y \end{bmatrix}} \tag{1.57}$$

$$\boxed{\|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{a}\| \, \|\mathbf{b}\| \sin\theta} \tag{1.58}$$

The cross product also exhibits the following properties (where $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ are vectors and k is an arbitrary constant).

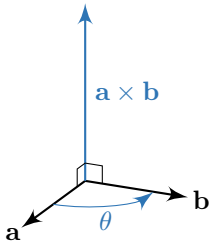


Figure 1.9: Cross product.

$$\boxed{\begin{aligned} \mathbf{a} \times \mathbf{b} &= -\mathbf{b} \times \mathbf{a} && \text{(1.59a)} \\ (k\mathbf{a}) \times \mathbf{b} &= k(\mathbf{a} \times \mathbf{b}) = \mathbf{a} \times (k\mathbf{b}) && \text{(1.59b)} \\ \mathbf{a} \times (\mathbf{b} + \mathbf{c}) &= \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c} && \text{(1.59c)} \\ (\mathbf{a} + \mathbf{b}) \times \mathbf{c} &= \mathbf{a} \times \mathbf{c} + \mathbf{b} \times \mathbf{c} && \text{(1.59d)} \\ \mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) &= (\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c} && \text{(1.59e)} \\ \mathbf{a} \times (\mathbf{b} \times \mathbf{c}) &= (\mathbf{a} \cdot \mathbf{c})\mathbf{b} - (\mathbf{a} \cdot \mathbf{b})\mathbf{c} && \text{(1.59f)} \end{aligned}}$$

Finally, we can also find the following identities for the cross products between the unit vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$.

$$\boxed{\begin{aligned} \hat{\mathbf{x}} \times \hat{\mathbf{y}} &= \hat{\mathbf{z}} && \text{(1.60a)} \\ \hat{\mathbf{y}} \times \hat{\mathbf{x}} &= -\hat{\mathbf{z}} && \text{(1.60b)} \\ \hat{\mathbf{y}} \times \hat{\mathbf{z}} &= \hat{\mathbf{x}} && \text{(1.60c)} \\ \hat{\mathbf{z}} \times \hat{\mathbf{y}} &= -\hat{\mathbf{x}} && \text{(1.60d)} \\ \hat{\mathbf{z}} \times \hat{\mathbf{x}} &= \hat{\mathbf{y}} && \text{(1.60e)} \\ \hat{\mathbf{x}} \times \hat{\mathbf{z}} &= -\hat{\mathbf{y}} && \text{(1.60f)} \end{aligned}}$$

Note that

$$\begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} b_z a_y - a_z b_y \\ b_x a_z - b_z a_x \\ b_y a_x - b_x a_y \end{bmatrix} \tag{1.61}$$

For a vector $\mathbf{a} = (a_x, a_y, a_z)^T$, let's define the **cross product matrix** $\mathbf{a}^\times$ as

$$\boxed{\mathbf{a}^\times = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}} \tag{1.62}$$

Then, the cross product between $\mathbf{a}$ and $\mathbf{b}$ can also be expressed as a matrix multiplication:

$$\boxed{\mathbf{a} \times \mathbf{b} = \mathbf{a}^\times \mathbf{b}} \tag{1.63}$$

Also note that [12, 45]

$$\boxed{\mathbf{a} \times \mathbf{b} = (\mathbf{b}^\times)^T \mathbf{a}} \tag{1.64}$$

Given $\mathbf{a}^\times = [a_{ij}^\times]$, we can also recover $\mathbf{a}$ as

$$\boxed{\mathbf{a} = \begin{bmatrix} a_{32}^\times \\ a_{13}^\times \\ a_{21}^\times \end{bmatrix}} \tag{1.65}$$

Algorithms 6 and 7 below implement Eqs. (1.65) and (1.62).

Algorithm 6: `vec2skew`

Converts a vector to a skew-symmetric matrix representing a cross product.

Inputs:

- $\mathbf{a} = (a_1, a_2, a_3)^T \in \mathbb{R}^3$ - vector

Procedure:

$$\mathbf{a}^\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

return $\mathbf{a}^\times$

Outputs:

- $\mathbf{a}^\times \in \mathbb{R}^{3\times 3}$ - skew-symmetric matrix where $\mathbf{a} \times \mathbf{b} = \mathbf{a}^\times \mathbf{b}$

Test Cases:

- See Appendix A.4.

Algorithm 7: `skew2vec`

Converts a skew-symmetric matrix representing a cross product to a vector.

Inputs:

- $\mathbf{a}^\times \in \mathbb{R}^{3\times 3}$ - skew-symmetric matrix where $\mathbf{a} \times \mathbf{b} = \mathbf{a}^\times \mathbf{b}$

Procedure:

$$\mathbf{a} = \begin{bmatrix} a_{32}^{\times} \\ a_{13}^{\times} \\ a_{21}^{\times} \end{bmatrix}$$

Outputs:

- $\mathbf{a} \in \mathbb{R}^3$ - vector

Test Cases:

- See Appendix A.4.

Orthogonal vectors

The vectors $\mathbf{a}$ and $\mathbf{b}$ are orthogonal if

$$\mathbf{a} \cdot \mathbf{b} = 0 \qquad \text{or} \qquad \frac{\|\mathbf{a} \times \mathbf{b}\|}{\|\mathbf{a}\| \, \|\mathbf{b}\|} = 1$$

Parallel vectors

The vectors $\mathbf{a}$ and $\mathbf{b}$ are parallel if

$$\mathbf{a} \times \mathbf{b} = \mathbf{0} \qquad \text{or} \qquad \|\mathbf{a} \times \mathbf{b}\| = 0 \qquad \text{or} \qquad \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|} = 1$$

Vector and scalar projections

The **vector projection** of $\mathbf{b}$ onto $\mathbf{a}$, denoted $\text{proj}_{\mathbf{a}}\mathbf{b}$, is a vector in the direction of $\mathbf{a}$ whose magnitude is the component of $\mathbf{b}$ in the direction of $\mathbf{a}$. The vector projection is given by

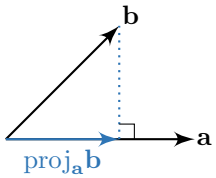


Figure 1.10: Vector projection.

$$\boxed{\text{proj}_{\mathbf{a}}\mathbf{b} = \frac{\mathbf{a}\,(\mathbf{a} \cdot \mathbf{b})}{\|\mathbf{a}\|^2} = \left(\frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|}\right)\frac{\mathbf{a}}{\|\mathbf{a}\|} = \left(\frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|}\right)\hat{\mathbf{a}} = \text{vector projection of } \mathbf{b} \text{ onto } \mathbf{a}} \tag{1.66}$$

The **scalar projection** of $\mathbf{b}$ onto $\mathbf{a}$ is a scalar defined as the magnitude of the vector projection of $\mathbf{b}$ onto $\mathbf{a}$, or as the magnitude of $\mathbf{b}$ in the direction of $\mathbf{a}$.

$$\text{comp}_{\mathbf{a}}\mathbf{b} = \left\| \left(\frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|}\right)\hat{\mathbf{a}} \right\| = \|\text{proj}_{\mathbf{a}}\mathbf{b}\| = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|}$$

$$\boxed{\text{comp}_{\mathbf{a}}\mathbf{b} = \frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|} = \|\text{proj}_{\mathbf{a}}\mathbf{b}\| = \text{scalar projection of } \mathbf{b} \text{ onto } \mathbf{a}} \tag{1.67}$$

Thus, we can also write the vector projection in terms of the scalar projection as

$$\boxed{\text{proj}_{\mathbf{a}}\mathbf{b} = (\text{comp}_{\mathbf{a}}\mathbf{b})\,\hat{\mathbf{a}} = \text{vector projection of } \mathbf{b} \text{ onto } \mathbf{a}} \tag{1.68}$$

Direction angles and direction cosines

The **direction angles** are the angles α, β, and γ, in the interval $[0, \pi]$, that $\mathbf{a}$ makes with the positive x-, y-, and z-axes, respectively. The **direction cosines** are the cosines of the direction angles and are given by

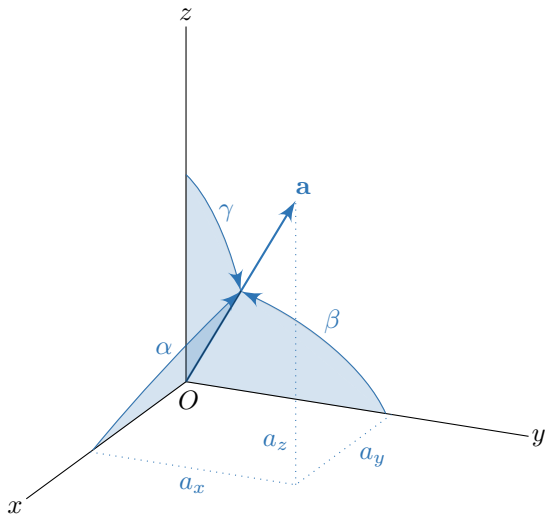


Figure 1.11: Direction angles.

$$\boxed{\cos\alpha = \frac{\mathbf{a}\cdot\hat{\mathbf{x}}}{\|\mathbf{a}\|\,\|\hat{\mathbf{x}}\|} = \frac{a_x}{\|\mathbf{a}\|} = \frac{a_x}{a} = \frac{a_x}{\sqrt{a_x^2+a_y^2+a_z^2}}} \tag{1.69}$$

$$\boxed{\cos\beta = \frac{\mathbf{a}\cdot\hat{\mathbf{y}}}{\|\mathbf{a}\|\,\|\hat{\mathbf{y}}\|} = \frac{a_y}{\|\mathbf{a}\|} = \frac{a_y}{a} = \frac{a_y}{\sqrt{a_x^2+a_y^2+a_z^2}}} \tag{1.70}$$

$$\boxed{\cos\gamma = \frac{\mathbf{a}\cdot\hat{\mathbf{z}}}{\|\mathbf{a}\|\,\|\hat{\mathbf{z}}\|} = \frac{a_z}{\|\mathbf{a}\|} = \frac{a_z}{a} = \frac{a_z}{\sqrt{a_x^2+a_y^2+a_z^2}}} \tag{1.71}$$

2

Physical Vectors

2.1 Physical Vectors and Matrices

Recall the column vector, $\mathbf{v} \in \mathbb{R}^n$, from Chapter 1. In the simplest sense, a column vector is just a list of numbers. For most of Chapter 1, we considered column vectors in coordinate systems (i.e. they were defined with respect to an ordered basis). However, there was no part of the vector itself that stored which coordinate system it was defined with respect to; it was simply a list of numbers. Thus, in the simplest sense, a list of numbers defines a column vector.

In dynamics, a **physical vector**[1] is an abstract quantity defined by some physical property; *not* by components or a coordinate system. More specifically, physical vectors describe physical quantities that have magnitude and direction in a three-dimensional space [20]. To describe our physical world, we attach coordinate systems to reference points. However, the physical world still exists without these coordinate systems, so physical vectors exist independent of coordinate systems as well, as shown in Fig. 2.1 below. This introduces some level of ambiguity, which allows

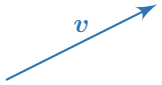


Figure 2.1: Physical vector.

us to write mathematical equations in a coordinate-free manner. The resulting equations are tantalizingly similar to matrix algebra, yet not quite the same. In most standard textbooks on dynamics and engineering, this distinction is either briefly mentioned or completely overlooked, as most examples and concepts can be taught without going into much depth. This becomes an issue when simulating complex physical systems using computers, which can only store column vectors. In this text, we emphasize this difference by devoting separate chapters to column and physical vectors. For an extremely clear, more in-depth discussion on this matter, see the introduction section of [44].

Note that in some sense, physical vectors *do* build upon column vectors; physical vectors can be **expressed** or **resolved** in a coordinate system. This will become more clear conceptually when we discuss coordinate vectors in Section 2.4.

[1] From this point on, we may refer to both column vectors and physical vectors simply as vectors. Using context clues, as well as the symbols used for the vectors, it should be clear which type of vector we are dealing with. In cases where we are using both physical vectors and column vectors, it will be made clear which is which.

Convention 12: Notation for physical vectors.

Physical vectors are written using lowercase, italic, boldface symbols. For example, a physical vector may be written as

$$\boldsymbol{v}$$

While a physical vector has magnitude and direction, a **physical scalar** only has a magnitude. They are similar to mathematical scalars in the sense that they can be represented using a single number. However, there is some ambiguity, since physical magnitudes also depend on what **units** they are expressed in.

Convention 13: Notation for physical scalars.

Physical scalars are written using lowercase or uppercase, italic, non-boldface symbols. For example, a physical scalar may be written as

$$a$$

The **magnitude** of a physical vector $\boldsymbol{v}$ is denoted as

$$\boxed{|\boldsymbol{v}| = v = \text{magnitude of } \boldsymbol{v}} \tag{2.1}$$

Convention 14: Notation for the magnitude of a physical vector.

Since the magnitude of a physical vector is a physical scalar, it is written using the italic, non-boldface version of the same symbol. For example, the magnitude of $\boldsymbol{v}$ is written as

$$v$$

To describe the **direction** of a physical vector, we use unit vectors. A **unit vector** is a vector of magnitude 1. Therefore, if we multiply a scalar quantity by a unit vector, we get a vector whose magnitude is equal to the scalar and whose direction is parallel to the unit vector's. The unit vector in the direction of $\boldsymbol{v}$ is defined as

$$\boxed{\hat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{|\boldsymbol{v}|} = \frac{\boldsymbol{v}}{v} = \text{direction of } \boldsymbol{v}} \tag{2.2}$$

Convention 15: Notation for physical unit vectors.

A physical unit vector in the direction of a physical vector is written using the same symbol but with a hat over it. For example, the unit vector in the direction of $\boldsymbol{v}$ is written as

$$\hat{\boldsymbol{v}}$$

By rearranging Eq. (2.2), we can write a vector $\boldsymbol{v}$ in terms of its magnitude and direction.

$$\boxed{\boldsymbol{v} = v\hat{\boldsymbol{v}}} \tag{2.3}$$

We mentioned that in some way, physical vectors are related to column vectors. Similarly, we can introduce the concept of a **physical matrix**, that is related to a mathematical matrix. The simplest example of a physical matrix is an inertia tensor[2], $\boldsymbol{I}$. An inertia tensor describing a three-dimensional body can be expressed in a three-dimensional coordinate system, resulting in a 3×3 mathematical matrix whose elements store the moments of inertia of the body with respect to the coordinate axes and planes.

Convention 16: Notation for physical matrices.

Physical matrices are written using uppercase, italic, boldface symbols. For example, a physical matrix may be written as

$$\boldsymbol{A}$$

2.2 Reference Frames

Just like a physical vector is an abstract notion of a column vector, a **reference frame** is an abstract notion of a coordinate system. It gives meaning to physical vectors in the sense that it defines what they are measured to.

As an example, consider a car that is traveling down the freeway. From the reference frame of a hitchhiker standing on the side of the freeway, the occupants of the car are moving. From the frame of reference of the car, the occupants are not moving; they remain in their seats, not moving relative to the car.

Note that in the example above, we didn't include any coordinate systems; this is because reference frames are completely independent of coordinate systems.

2.3 Coordinate Frames

Coordinate frames are the link between coordinate systems and reference frames. Coordinate systems can exist completely independently of reference frames, and reference frames can exist completely independently of coordinate systems. Coordinate frames are the special case where we attach a coordinate system *to* a reference frame in order to quantify vectors with respect to that reference frame. Essentially,

$$(\text{coordinate system}) + (\text{reference frame}) = (\text{coordinate frame})$$

Note that in a coordinate frame, the coordinate system remains fixed to the reference frame; it can never move relative to the reference frame it describes.

Whereas a coordinate system is defined by an ordered basis consisting of *mathematical* basis vector, a coordinate frame is defined by an ordered basis consisting of *physical* basis vectors. For example, the coordinate frame describe the $Oxyz$ coordinate system would have the basis $A = (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$. Note that in this case, the unit (basis) vectors themselves are also physical vectors; they are not simply lists of numbers. In the case of column vectors, the coordinate systems were complete abstractions, so we could set the basis vectors to be sets of numbers with no relation to the physical world. However, in the case of physical vectors, the basis vectors *do* often define extremely specific directions in our physical world[3]

[2] In mathematics, a tensor is generalization of a matrix to higher-dimensional vector spaces. In dynamics, an inertia tensor colloquially refers to a mathematical matrix storing the moments of inertia of a three-dimensional body. Note that the inertia tensor is also sometimes referred to simply as the inertia *matrix* [29, 51].

[3] As an example, the coordinate system for the Earth-centered inertial coordinate frame is defined such that its X axis points towards the vernal equinox.

2.4 Coordinate Vectors

Just how coordinate frames are the link between coordinate systems and reference frames, **coordinate vectors** are the link between column vectors and physical vectors. Consider a physical vector, $\boldsymbol{v}$. Now, consider a coordinate frame $A = (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$ defining the coordinate system $Oxyz$. Similar to how column vectors could be written in terms of the basis vectors of a coordinate *system* in Eq. (1.8), we can write physical vectors in terms of the basis vectors of a coordinate *frame*.

$$\boxed{\boldsymbol{v} = v_x\hat{\boldsymbol{x}} + v_y\hat{\boldsymbol{y}} + v_z\hat{\boldsymbol{z}}} \tag{2.4}$$

v_x, v_y, and v_z are the **coordinates** or **components** of $\boldsymbol{v}$.

As an aside, we can define the magnitude, v, and direction, $\hat{\boldsymbol{v}}$, in terms of its components.

$$\boxed{v = \sqrt{v_x^2 + v_y^2 + v_z^2}} \tag{2.5}$$

$$\boxed{\hat{\boldsymbol{v}} = \left(\frac{v_x}{v}\right)\hat{\boldsymbol{x}} + \left(\frac{v_y}{v}\right)\hat{\boldsymbol{y}} + \left(\frac{v_z}{v}\right)\hat{\boldsymbol{z}}} \tag{2.6}$$

Let's rewrite Eq. (2.4) using matrix algebra.

$$\boldsymbol{v} = \underbrace{\begin{bmatrix}\hat{\boldsymbol{x}} & \hat{\boldsymbol{y}} & \hat{\boldsymbol{z}}\end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix}v_x \\ v_y \\ v_z\end{bmatrix}}_{[\boldsymbol{v}]_A}$$

We define the **frame matrix** as

$$\boxed{\boldsymbol{A} = \begin{bmatrix}\hat{\boldsymbol{x}} & \hat{\boldsymbol{y}} & \hat{\boldsymbol{z}}\end{bmatrix}} \tag{2.7}$$

and the **coordinate vector** of $\boldsymbol{v}$ when expressed in coordinate frame A as [11], [26, pp. 163–172]

$$\boxed{[\boldsymbol{v}]_A = \begin{bmatrix}v_x \\ v_y \\ v_z\end{bmatrix}} \tag{2.8}$$

Convention 17: Coordinate vector notation.

A physical vector expressed in a coordinate frame should be placed in brackets and subscripted (outside the brackets) with the symbol representing the coordinate frame. For example, a physical vector, $\boldsymbol{v}$, expressed in the coordinate frame A should be denoted

$$[\boldsymbol{v}]_A$$

If the vector has a descriptive subscript, the descriptive subscript remains inside the brackets, while the frame subscript is outside the brackets. For example, if a vector has the descriptive subscript "ex", the corresponding coordinate vector expressed in the coordinate frame A is

$$[\boldsymbol{v}_{\text{ex}}]_A$$

We can write $\boldsymbol{v}$ in terms of $\boldsymbol{A}$ and $[\boldsymbol{v}]_A$ as

$$\boxed{\boldsymbol{v} = \boldsymbol{A}[\boldsymbol{v}]_A} \tag{2.9}$$

Note that a frame matrix can also be expressed in a coordinate frame. For example, if we have another coordinate frame $B = (\hat{\mathbf{a}}, \hat{\mathbf{b}}, \hat{\mathbf{c}})$,

$$\boxed{[\boldsymbol{A}]_B = \begin{pmatrix}[\hat{\boldsymbol{x}}]_B & [\hat{\boldsymbol{y}}]_B & [\hat{\boldsymbol{z}}]_B\end{pmatrix}} \tag{2.10}$$

2.5 Frame Transformation via Rotation

Consider the physical vector $\boldsymbol{v}$ and the coordinate frames $A = (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$ and $B = (\hat{\boldsymbol{a}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{v}})$. Using Eq. (2.9), we express $\boldsymbol{v}$ in either coordinate frame.

$$\boldsymbol{v} = \boldsymbol{A}[\boldsymbol{v}]_A = \boldsymbol{B}[\boldsymbol{v}]_B$$

In the equation above, $\boldsymbol{A}$ and $\boldsymbol{B}$ are the frame matrices given by

$$\boldsymbol{A} = \begin{bmatrix} \hat{\boldsymbol{x}} & \hat{\boldsymbol{y}} & \hat{\boldsymbol{z}} \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} \hat{\boldsymbol{a}} & \hat{\boldsymbol{b}} & \hat{\boldsymbol{c}} \end{bmatrix}$$

while $[\boldsymbol{v}]_A$ and $[\boldsymbol{v}]_B$ are the coordinate vectors of $\boldsymbol{v}$ expressed in bases A and B, respectively.

$$[\boldsymbol{v}]_A = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}, \qquad [\boldsymbol{v}]_B = \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix}$$

Consider the case where we have knowledge of the basis vectors of A expressed in coordinate frame B. Then we have the coordinate matrix

$$[\boldsymbol{A}]_B = \begin{pmatrix} [\hat{\boldsymbol{x}}]_B & [\hat{\boldsymbol{y}}]_B & [\hat{\boldsymbol{z}}]_B \end{pmatrix}$$

We know from Eq. (1.20) that

$$[\boldsymbol{v}]_B = [\boldsymbol{A}]_B [\boldsymbol{v}]_A$$

Furthermore, we have $[\boldsymbol{v}]_A, [\boldsymbol{v}]_B \in \mathbb{R}^3$ and $[\boldsymbol{A}]_B \in \mathbb{R}^{3\times 3}$, so $[\boldsymbol{A}]_B$ represents a rotation matrix (see Section 1.6). Thus, we write

$$\boxed{[\boldsymbol{v}]_A = \mathbf{R}_{A\to B}[\boldsymbol{v}]_B} \tag{2.11}$$

where $\mathbf{R}_{A\to B}$ is the rotation matrix from coordinate frame A to coordinate frame B.

Convention 18: Notation for a rotation matrix between physical bases.

Consider the physical vector $\boldsymbol{v}$ and the coordinate frames $A = (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$ and $B = (\hat{\boldsymbol{a}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{v}})$. The rotation matrix from coordinate frame A to coordinate frame B is written as

$$\mathbf{R}_{A\to B}$$

and satisfies

$$[\boldsymbol{v}]_B = \mathbf{R}_{A\to B}[\boldsymbol{v}]_A$$

Algorithm 8: `matrotate`
Passive rotation of a vector by a rotation matrix.

Inputs:
- $[\mathbf{R}_{A\to B} \in \mathbb{R}^{3\times 3}$ - rotation matrix representing rotation from frame A to frame B
- $[\boldsymbol{r}]_A \in \mathbb{R}^3$ - vector expressed in coordinate frame A

Procedure:

$[\boldsymbol{r}]_B = \mathbf{R}_{A\to B}[\boldsymbol{r}]_A$
return $[\boldsymbol{r}]_B$

Outputs:
- $[\boldsymbol{r}]_B \in \mathbb{R}^3$ - vector expressed in coordinate frame B

Test Cases:
- See Appendix A.1.1.

If we wanted to solve for $[\boldsymbol{v}]_B$ given $[\boldsymbol{v}]_A$, we could left-multiply both sides by $\mathbf{R}_{A\to B}^T$ (since the inverse of a rotation matrix is its transpose).

$$[\boldsymbol{v}]_A = \mathbf{R}_{A\to B}^T [\boldsymbol{v}]_B$$

However, note that

$$[\boldsymbol{v}]_A = \mathbf{R}_{B\to A} [\boldsymbol{v}]_B$$

Thus, the rotation matrix for the inverse transformation is

$$\boxed{\mathbf{R}_{B\to A} = \mathbf{R}_{A\to B}^T} \tag{2.12}$$

2.5.1 Chained Rotations

Let $\mathbf{R}_{A\to B}$ be the rotation matrix from frame A to frame B, and let $\mathbf{R}_{B\to C}$ be the rotation matrix from frame B to frame C. Let's start with a physical vector, $\boldsymbol{r}$, expressed in frame A, and find the coordinates of that vector expressed in frame B.

$$[\boldsymbol{r}]_B = \mathbf{R}_{A\to B} [\boldsymbol{r}]_A \tag{2.13}$$

Now that we know $[\boldsymbol{r}]_B$, we can find $[\boldsymbol{r}]_C$ using $\mathbf{R}_{B\to C}$.

$$[\boldsymbol{r}]_C = \mathbf{R}_{B\to C} [\boldsymbol{r}]_B \tag{2.14}$$

Our goal is to perform this **chained rotation** in a single step as

$$[\boldsymbol{r}]_C = \mathbf{R}_{A\to C} [\boldsymbol{r}]_A$$

Substituting Eq. (2.13) into Eq. (2.14),

$$\begin{aligned} [\boldsymbol{r}]_C &= \mathbf{R}_{B\to C} \left(\mathbf{R}_{A\to B} [\boldsymbol{r}]_A \right) \\ &= \underbrace{(\mathbf{R}_{B\to C} \mathbf{R}_{A\to B})}_{\mathbf{R}_{A\to C}} [\boldsymbol{r}]_A \end{aligned}$$

Thus, we have the following rule for chaining together rotation matrices:

$$\boxed{\mathbf{R}_{A\to C} = \mathbf{R}_{B\to C} \mathbf{R}_{A\to B}} \tag{2.15}$$

This chaining can be extended indefinitely. For example,

$$\mathbf{R}_{A\to E} = \mathbf{R}_{D\to E} \mathbf{R}_{C\to D} \mathbf{R}_{B\to C} \mathbf{R}_{A\to B}$$

Algorithm 9: `matchain`
Chaining passive rotations represented by rotation matrices.

Inputs:
- $\mathbf{R}_{A\to B} \in \mathbb{R}^{3\times 3}$ - rotation matrix representing rotation from frame A to frame B
- $\mathbf{R}_{B\to C} \in \mathbb{R}^{3\times 3}$ - rotation matrix representing rotation from frame B to frame C

Procedure:

$\mathbf{R}_{A\to C} = \mathbf{R}_{B\to C} \mathbf{R}_{A\to B}$

return $\mathbf{R}_{A\to C}$

Outputs:
- $\mathbf{R}_{A\to C} \in \mathbb{R}^{3\times 3}$ - rotation matrix representing rotation from frame A to frame C

Test Cases:
- See Appendix A.1.1.

2.5.2 Constructing Rotation Matrices from Basis Vectors

Earlier, we skimmed over the fact that the rotation matrix from coordinate frame A to coordinate frame B is simply

$$\mathbf{R}_{A\to B} = [\boldsymbol{A}]_B$$

where

$$[\boldsymbol{A}]_B = \begin{pmatrix} [\hat{\boldsymbol{x}}]_B & [\hat{\boldsymbol{y}}]_B & [\hat{\boldsymbol{z}}]_B \end{pmatrix}$$

is the coordinate matrix of coordinate frame A expressed in coordinate frame B. Thus, if we know the unit vectors of one frame expressed in the other frame, then we can form a rotation matrix between the two frames.

$$\boxed{\mathbf{R}_{A\to B} = \begin{bmatrix} [\hat{\boldsymbol{x}}]_B & [\hat{\boldsymbol{y}}]_B & [\hat{\boldsymbol{z}}]_B \end{bmatrix} \qquad ([\hat{\boldsymbol{x}}]_B, [\hat{\boldsymbol{y}}]_B, \text{and } [\hat{\boldsymbol{z}}]_B \text{ are the basis vectors of } A \text{ expressed in } B)} \tag{2.16}$$

Conversely, we also know that

$$\mathbf{R}_{B\to A} = \begin{pmatrix} [\hat{\boldsymbol{a}}]_A & [\hat{\boldsymbol{b}}]_A & [\hat{\boldsymbol{c}}]_A \end{pmatrix}$$

where $[\hat{\boldsymbol{a}}]_A$, $[\hat{\boldsymbol{b}}]_A$, and $[\hat{\boldsymbol{c}}]_A$ are the basis vectors of frame B expressed in frame A. Since $\mathbf{R}_{A\to B} = \mathbf{R}_{B\to A}^T$,

$$\mathbf{R}_{A\to B} = \mathbf{R}_{B\to A}^T = \begin{pmatrix} [\hat{\boldsymbol{a}}]_A & [\hat{\boldsymbol{b}}]_A & [\hat{\boldsymbol{c}}]_A \end{pmatrix}^T$$

$$\boxed{\mathbf{R}_{A\to B} = \begin{pmatrix} [\hat{\boldsymbol{a}}]_A^T \\ [\hat{\boldsymbol{b}}]_A^T \\ [\hat{\boldsymbol{c}}]_A^T \end{pmatrix} \qquad ([\hat{\boldsymbol{a}}]_A, [\hat{\boldsymbol{b}}]_A, \text{and } [\hat{\boldsymbol{c}}]_A \text{ are the basis vectors of } B \text{ expressed in } A)} \tag{2.17}$$

2.6 The Algebra of Physical Vectors

In Section 2.1, we described how we often write physics/dynamics equations in terms of physical vectors, but in practice (especially in simulation) we need to write them in terms of coordinate vectors that we can actually perform arithmetic with. Let's consider the simplest possible equation below, where we add two physical vectors.

$$\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b}$$

Now, let's introduce a coordinate frame, F. We can write this equation in terms of the coordinate vectors expressed in frame F by using the frame matrix, $\boldsymbol{F}$, corresponding to frame F (see Section 2.4).

$$\boldsymbol{F}[\boldsymbol{c}]_F = \boldsymbol{F}[\boldsymbol{a}]_F + \boldsymbol{F}[\boldsymbol{b}]_F \tag{2.18}$$

Recall that a frame matrix is internally composed of three physical basis vectors. In this case,

$$\boldsymbol{F} = \begin{bmatrix} \hat{\boldsymbol{x}} & \hat{\boldsymbol{y}} & \hat{\boldsymbol{z}} \end{bmatrix}$$

This is analogous to the case of writing a 3×3 mathematical matrix $\mathbf{A} \in \mathbb{R}^{3\times 3}$ in terms of its three column vectors, $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3 \in \mathbb{R}^3$.

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{bmatrix}$$

Since we can perform any matrix algebra in terms of partitioned matrices[4] (in this case, we are partitioning the matrix into its columns), we can just interpret a frame matrix as a partitioned matrix (this time partitioned into three physical vectors). Although the blocks of the frame matrix are abstract quantities, we can just treat them as any other vector. Thus, continuing our example from Eq. (2.18),

$$\boldsymbol{F}[\boldsymbol{c}]_F = \boldsymbol{F}\left([\boldsymbol{a}]_F + [\boldsymbol{b}]_F\right)$$

This implies that

$$[\boldsymbol{c}]_F = [\boldsymbol{a}]_F + [\boldsymbol{b}]_F$$

Recall the equation we started with at the beginning of this section. We have essentially showed that the coordinate vector form of an equation is largely analogous to the physical vector form.

$$\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b} \quad \Longleftrightarrow \quad [\boldsymbol{c}]_F = [\boldsymbol{a}]_F + [\boldsymbol{b}]_F$$

The algebra of physical vectors and matrices is essentially the same as the algebra of mathematical vectors and matrices.

Take this statement with a grain of salt; we did no rigorous proofs here and merely presented the simplest example. There are likely cases where this statement could break down. While we could easily go more in-depth, this is enough to demonstrate how the algebra of physical vectors and matrices mimics that of mathematical vectors and matrices. This statement is applicable throughout most of the applications in this text.

2.6.1 Substituting Coordinate and Physical Vectors

Consider an arbitrary physical vector $\boldsymbol{v}$ and the coordinate frames A and B (with frame matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively). Recall that we can write $\boldsymbol{v}$ in terms of its coordinates vectors expressed in either of these frames as

$$\boldsymbol{v} = \boldsymbol{A}[\boldsymbol{v}]_A = \boldsymbol{B}[\boldsymbol{v}]_B \tag{2.19}$$

Also recall that we can write the coordinate vectors in terms of one another using the rotation matrix between the two frames.

$$[\boldsymbol{v}]_B = \mathbf{R}_{A\to B}[\boldsymbol{v}]_A$$

Substituting this into the first equation, we have

$$\boldsymbol{v} = \boldsymbol{A}[\boldsymbol{v}]_A = \boldsymbol{B}\mathbf{R}_{A\to B}[\boldsymbol{v}]_A$$

Thus, we can always make the following substitutions:

$$\boxed{\begin{aligned}\boldsymbol{v} &= \boldsymbol{A}[\boldsymbol{v}]_A \\ \boldsymbol{v} &= \boldsymbol{B}\mathbf{R}_{A\to B}[\boldsymbol{v}]_A\end{aligned}} \tag{2.20}$$

By comparing the two lines of Eq. (2.20), we can also see that

$$\boxed{\boldsymbol{A} = \boldsymbol{B}\mathbf{R}_{A\to B}} \tag{2.21}$$

While we easily arrived at Eq. (2.20) above, we extend the general idea provided by this equation below, but provide no proof. Instead, we will provide an example that should illustrate the concept, but nonetheless we concede that this statement can appear quite "hand-wavy".

[4] See https://en.wikipedia.org/wiki/Block_matrix.

In Eq. (2.20), if $[\boldsymbol{v}]_A$ is a coordinate vector expressed in frame A that is the result of some vector operations on coordinate vectors, then $\boldsymbol{v}$ can be defined using that same sequence of operations but on the *physical* vectors that the coordinate vectors describe.

The statement above is best illustrated through an example. Consider three coordinate frames, A, B, and C, and three coordinate vectors $[\boldsymbol{x}]_A$, $[\boldsymbol{y}]_B$, $[\boldsymbol{z}]_C$. Say we have the expression

$$\boldsymbol{B}\left[(\mathbf{R}_{A\to B}[\boldsymbol{x}]_A)\times([\boldsymbol{y}]_B)\times(\mathbf{R}_{C\to B}[\boldsymbol{z}]_C)\right]$$

Since $\mathbf{R}_{A\to B}[\boldsymbol{x}]_A$, $[\boldsymbol{y}]_B$, and $\mathbf{R}_{C\to B}[\boldsymbol{z}]_C$ are all coordinate vectors expressed in frame B, the result of the triple cross product is also a coordinate vector expressed in frame B. Furthermore, since we are multiplying by the frame matrix $\boldsymbol{B}$, Eq. (2.20) implies

$$\boldsymbol{x}\times\boldsymbol{y}\times\boldsymbol{z}=\boldsymbol{B}\left[(\mathbf{R}_{A\to B}[\boldsymbol{x}]_A)\times([\boldsymbol{y}]_B)\times(\mathbf{R}_{C\to B}[\boldsymbol{z}]_C)\right]$$

In Section 3.3.2, we will use these substitutions to convert an equation involving coordinate vectors to an equation expressed using only physical vectors. However, in most cases, we will be taking the opposite route; we will have some physical law written in terms of physical vectors that we will want to express in terms of coordinate vectors in order to be able to implement in a simulation.

2.7 Physical Vector Operations

In the previous section, we discussed how the algebra of physical vectors mimics that of mathematical vectors. Additionally, since physical vectors are tied to a three-dimensional world, we can define some of the same operations for physical vectors as we did for mathematical vectors in Section 1.8.

Dot (scalar) product

The **dot (scalar) product** between two physical vectors, $\boldsymbol{a}$ and $\boldsymbol{b}$, is defined as

$$\boxed{\boldsymbol{a}\cdot\boldsymbol{b}=|\boldsymbol{a}|\,|\boldsymbol{b}|\cos\theta} \tag{2.22}$$

where θ is the angle between the two vectors. Note that regardless of which coordinate frames the vectors are defined in, there *always* exists some angle θ between them.

It follows that the dot product is also frame-independent; the magnitudes of $\boldsymbol{a}$ and $\boldsymbol{b}$ are already independent of any coordinate frame since they describe the *magnitude* of the vectors, not their *direction*.

The dot product also exhibits the following properties (where $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ are physical vectors and k is an arbitrary constant).

$$\boldsymbol{a}\cdot\boldsymbol{a}=|\boldsymbol{a}|^2 \tag{2.23a}$$

$$\boldsymbol{a}\cdot(\boldsymbol{b}+\boldsymbol{c})=\boldsymbol{a}\cdot\boldsymbol{b}+\boldsymbol{a}\cdot\boldsymbol{c} \tag{2.23b}$$

$$\mathbf{0}\cdot\boldsymbol{a}=0 \tag{2.23c}$$

$$\boldsymbol{a}\cdot\boldsymbol{b}=\boldsymbol{b}\cdot\boldsymbol{a} \tag{2.23d}$$

$$(k\boldsymbol{a})\cdot\boldsymbol{b}=k(\boldsymbol{a}\cdot\boldsymbol{b})=\boldsymbol{a}\cdot(k\boldsymbol{b}) \tag{2.23e}$$

Finally, we can also find the following identities for the dot products between the unit vectors $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$, and $\hat{\boldsymbol{z}}$.

$$\hat{\boldsymbol{x}} \cdot \hat{\boldsymbol{y}} = \hat{\boldsymbol{y}} \cdot \hat{\boldsymbol{z}} = \hat{\boldsymbol{z}} \cdot \hat{\boldsymbol{x}} = 0 \tag{2.24a}$$

$$\hat{\boldsymbol{x}} \cdot \hat{\boldsymbol{x}} = \hat{\boldsymbol{y}} \cdot \hat{\boldsymbol{y}} = \hat{\boldsymbol{z}} \cdot \hat{\boldsymbol{z}} = 1 \tag{2.24b}$$

Cross (vector) product

The **cross (vector) product** of $\boldsymbol{a}$ and $\boldsymbol{b}$ is a vector multiplication that produces a third vector that is normal/orthogonal/perpendicular to both $\boldsymbol{a}$ and $\boldsymbol{b}$.

$$|\boldsymbol{a} \times \boldsymbol{b}| = |\boldsymbol{a}|\,|\boldsymbol{b}| \sin\theta \tag{2.25}$$

The cross product also exhibits the following properties (where $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ are vectors and k is an arbitrary constant).

$$\boldsymbol{a} \times \boldsymbol{b} = -\boldsymbol{b} \times \boldsymbol{a} \tag{2.26a}$$

$$(k\boldsymbol{a}) \times \boldsymbol{b} = k(\boldsymbol{a} \times \boldsymbol{b}) = \boldsymbol{a} \times (k\boldsymbol{b}) \tag{2.26b}$$

$$\boldsymbol{a} \times (\boldsymbol{b} + \boldsymbol{c}) = \boldsymbol{a} \times \boldsymbol{b} + \boldsymbol{a} \times \boldsymbol{c} \tag{2.26c}$$

$$(\boldsymbol{a} + \boldsymbol{b}) \times \boldsymbol{c} = \boldsymbol{a} \times \boldsymbol{c} + \boldsymbol{b} \times \boldsymbol{c} \tag{2.26d}$$

$$\boldsymbol{a} \cdot (\boldsymbol{b} \times \boldsymbol{c}) = (\boldsymbol{a} \times \boldsymbol{b}) \cdot \boldsymbol{c} \tag{2.26e}$$

$$\boldsymbol{a} \times (\boldsymbol{b} \times \boldsymbol{c}) = (\boldsymbol{a} \cdot \boldsymbol{c})\boldsymbol{b} - (\boldsymbol{a} \cdot \boldsymbol{b})\boldsymbol{c} \tag{2.26f}$$

Finally, we can also find the following identities for the cross products between the unit vectors $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$, and $\hat{\boldsymbol{z}}$.

$$\hat{\boldsymbol{x}} \times \hat{\boldsymbol{y}} = \hat{\boldsymbol{z}} \tag{2.27a}$$

$$\hat{\boldsymbol{y}} \times \hat{\boldsymbol{x}} = -\hat{\boldsymbol{z}} \tag{2.27b}$$

$$\hat{\boldsymbol{y}} \times \hat{\boldsymbol{z}} = \hat{\boldsymbol{x}} \tag{2.27c}$$

$$\hat{\boldsymbol{z}} \times \hat{\boldsymbol{y}} = -\hat{\boldsymbol{x}} \tag{2.27d}$$

$$\hat{\boldsymbol{z}} \times \hat{\boldsymbol{x}} = \hat{\boldsymbol{y}} \tag{2.27e}$$

$$\hat{\boldsymbol{x}} \times \hat{\boldsymbol{z}} = -\hat{\boldsymbol{y}} \tag{2.27f}$$

Orthogonal vectors

The vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ are orthogonal if

$$\boldsymbol{a} \cdot \boldsymbol{b} = 0 \qquad \text{or} \qquad \frac{|\boldsymbol{a} \times \boldsymbol{b}|}{|\boldsymbol{a}|\,|\boldsymbol{b}|} = 1$$

Parallel vectors

The vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ are parallel if

$$\boldsymbol{a} \times \boldsymbol{b} = \mathbf{0} \qquad \text{or} \quad |\boldsymbol{a} \times \boldsymbol{b}| = 0 \qquad \text{or} \qquad \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{|\boldsymbol{a}|\,|\boldsymbol{b}|} = 1$$

3

Translational Kinematics

TODO: redo so rate vector is defined during the derivative of the coordinate vector

3.1 Derivatives of Mathematical Vectors

Consider a mathematical vector, $\mathbf{v} \in \mathbb{R}^n$.

$$\mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

Let's say that $\mathbf{v}$ is a function of a scalar independent variable, $t \in \mathbb{R}$.

$$\mathbf{v} = \begin{bmatrix} v_1(t) \\ \vdots \\ v_n(t) \end{bmatrix}$$

Then the derivative of $\mathbf{v}$ with respect to t is

$$\boxed{\frac{d\mathbf{v}}{dt} = \begin{bmatrix} \frac{dv_1}{dt} \\ \vdots \\ \frac{dv_n}{dt} \end{bmatrix} \in \mathbb{R}^n} \tag{3.1}$$

3.2 Derivatives of Coordinate Vectors

Consider the coordinate vectors $[\mathbf{v}]_X \in \mathbb{R}^n$ and $[\mathbf{v}]_Y \in \mathbb{R}^n$, which represent the same vector expressed in two separate coordinate systems with the same origin, $Ox_1...x_n$ and $Oy_1...y_n$, respectively. These coordinate systems are defined

by the ordered bases $X = (\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_n)$ and $Y = (\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n)$, respectively.

$$[\mathbf{v}]_X = \begin{bmatrix} v_{x_1} \\ \vdots \\ v_{x_n} \end{bmatrix}, \qquad [\mathbf{v}]_Y = \begin{bmatrix} v_{y_1} \\ \vdots \\ v_{y_n} \end{bmatrix}$$

Allowing them to vary as univariate functions of a scalar variable, $t \in \mathbb{R}$,

$$[\mathbf{v}]_X(t) = \begin{bmatrix} v_{x_1}(t) \\ \vdots \\ v_{x_n}(t) \end{bmatrix}, \qquad [\mathbf{v}]_Y(t) = \begin{bmatrix} v_{y_1}(t) \\ \vdots \\ v_{y_n}(t) \end{bmatrix}$$

If we know both $[\mathbf{v}]_X$ and $[\mathbf{v}]_Y$, then their derivatives with respect to a scalar independent variable, $t \in \mathbb{R}$, are simply

$$\boxed{\frac{d[\mathbf{v}]_X}{dt} = \begin{bmatrix} \frac{dv_{x_1}}{dt} \\ \vdots \\ \frac{dv_{x_n}}{dt} \end{bmatrix} \in \mathbb{R}^n, \qquad \frac{d[\mathbf{v}]_Y}{dt} = \begin{bmatrix} \frac{dv_{y_1}}{dt} \\ \vdots \\ \frac{dv_{y_n}}{dt} \end{bmatrix} \in \mathbb{R}^n} \tag{3.2}$$

Now, let's consider the case where we know $[\mathbf{v}]_X(t)$ and $[\mathbf{X}]_Y(t)$, and want to find the derivative of $[\mathbf{v}]_Y(t)$ with respect to t. Note that

$$[\mathbf{X}]_Y(t) = \begin{pmatrix} [\hat{\mathbf{x}}_1]_Y & \cdots & [\hat{\mathbf{x}}_n]_Y \end{pmatrix}$$

is that basis matrix of X expressed in the basis Y. From Section 1.4, we know we can write $[\mathbf{v}]_Y$ in terms of $[\mathbf{v}]_X$ as

$$[\mathbf{v}]_Y = [\mathbf{X}]_Y [\mathbf{v}]_X$$

Differentiating both sides with respect to t,

$$\boxed{\frac{d[\mathbf{v}]_Y}{dt} = \left(\frac{d[\mathbf{X}]_Y}{dt}\right)[\mathbf{v}]_X + [\mathbf{X}]_Y \left(\frac{d[\mathbf{v}]_X}{dt}\right)} \tag{3.3}$$

3.2.1 Orthogonal Bases

If X and Y are both *orthogonal* bases, then we know that

$$[\mathbf{X}]_Y^T [\mathbf{X}]_Y = \mathbf{I}_{n\times n}$$

Differentiating both sides with respect to t,

$$\left(\frac{d[\mathbf{X}]_Y}{dt}\right)[\mathbf{X}]_Y^T + [\mathbf{X}]_Y \left(\frac{d[\mathbf{X}]_Y^T}{dt}\right) = \mathbf{0}_{n\times n}$$

Noting that

$$\frac{d\mathbf{M}^T}{dt} = \left(\frac{d\mathbf{M}}{dt}\right)^T$$

for an arbitrary matrix $\mathbf{M}$, we have

$$\left(\frac{d[\mathbf{X}]_Y}{dt}\right)[\mathbf{X}]_Y^T + [\mathbf{X}]_Y \left(\frac{d[\mathbf{X}]_Y}{dt}\right)^T = \mathbf{0}_{n\times n} \tag{3.4}$$

Now, let's define

$$\boxed{[\mathbf{W}]_Y = \left(\frac{d[\mathbf{X}]_Y}{dt}\right)[\mathbf{X}]_Y^T} \tag{3.5}$$

We use a subscript Y because everything on the right hand side is expressed in Y, so everything on the left hand side (i.e. $[\mathbf{W}]_Y$) must be expressed in Y as well.

Solving for $d[\mathbf{X}]_Y/dt$,

$$\frac{d[\mathbf{X}]_Y}{dt} = [\mathbf{W}]_Y \left([\mathbf{X}]_Y^T\right)^{-1}$$

Since $[\mathbf{X}]_Y$ is an orthogonal matrix, $[\mathbf{X}]_Y^{-1} = [\mathbf{X}]_Y^T$, so we have [58]

$$\boxed{\frac{d[\mathbf{X}]_Y}{dt} = [\mathbf{W}]_Y[\mathbf{X}]_Y} \tag{3.6}$$

In general, we do not know $d[\mathbf{X}]_Y/dt$, i.e. we do not know the derivatives of the basis vectors of X expressed in Y, However, for the case of *orthogonal matrices*, Eq (3.6) allows us to obtain this derivative using two quantities measured at t instead: $[\mathbf{X}]_Y(t)$ and $[\mathbf{W}]_Y(t)$.

Substituting Eq. (**??**) into Eq. (3.4),

$$\begin{aligned}
\mathbf{0}_{n\times n} &= \left([\mathbf{W}]_Y[\mathbf{X}]_Y\right)[\mathbf{X}]_Y^T + [\mathbf{X}]_Y\left([\mathbf{W}]_Y[\mathbf{X}]_Y\right)^T \\
&= [\mathbf{W}]_Y\left([\mathbf{X}]_Y[\mathbf{X}]_Y^T\right) + [\mathbf{X}]_Y\left([\mathbf{X}]_Y^T[\mathbf{W}]_Y^T\right) \\
&\quad [\mathbf{W}]_Y\left([\mathbf{X}]_Y[\mathbf{X}]_Y^T\right) + \left([\mathbf{X}]_Y[\mathbf{X}]_Y^T\right)[\mathbf{W}]_Y^T \\
&= [\mathbf{W}]_Y\left(\mathbf{I}_{n\times n}\right) + \left(\mathbf{I}_{n\times n}\right)[\mathbf{W}]_Y^T
\end{aligned}$$

$$\boxed{[\mathbf{W}]_Y + [\mathbf{W}]_Y^T = \mathbf{0}_{n\times n}} \tag{3.7}$$

Eq. (3.7) implies that $[\mathbf{W}]_Y$ is a skew-symmetric matrix[1]; this is an essential property of $[\mathbf{W}]_Y$ that is exploited in the three-dimensional case in Section 3.2.1. Now, substituting Eq. (3.6) into Eq. (3.3),

$$\frac{d[\mathbf{v}]_Y}{dt} = \left([\mathbf{W}]_Y[\mathbf{X}]_Y\right)[\mathbf{v}]_X + [\mathbf{X}]_Y\left(\frac{d[\mathbf{v}]_X}{dt}\right)$$

$$\boxed{\frac{d[\mathbf{v}]_Y}{dt} = [\mathbf{X}]_Y\left(\frac{d[\mathbf{v}]_X}{dt}\right) + [\mathbf{W}]_Y[\mathbf{X}]_Y[\mathbf{v}]_X} \tag{3.8}$$

3.3 Derivatives of Physical Vectors

TODO: go through all sections, ensure that it is clearly labelled that coordinate systems have same origin

Consider a physical vector, $\boldsymbol{v}$. Its coordinate vector expressed in some three-dimensional basis $A = (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$ is $\mathbf{v}_A \in \mathbb{R}^3$. Our goal is to find its derivative with respect to another three-dimensional basis $B = (\hat{\boldsymbol{a}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{c}})$ that is rotating with respect to A.

3.3.1 Derivative of the Coordinate Vector

Recall from Eq. 3.8 in Section 3.2.1 that the derivative of an n-dimensional coordinate vector expressed in frame A can be written in terms of the coordinate vector (and its derivative) expressed in frame B as

$$\frac{d[\boldsymbol{v}]_B}{dt} = \mathbf{A}_B\left(\frac{d[\boldsymbol{v}]_A}{dt}\right) + [\mathbf{W}]_B\mathbf{A}_B[\boldsymbol{v}]_A$$

[1] See Section 1.7.

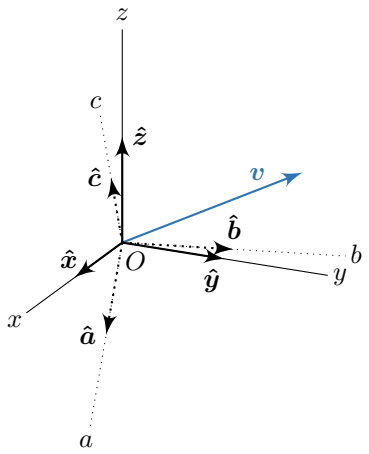


Figure 3.1: Physical vector with respect to two coordinate frames.

where $\mathbf{A}_B$ is the basis matrix of A expressed in B and $[\mathbf{W}]_B$ is some matrix expressed in B that satisfies

$$[\mathbf{W}]_B + [\mathbf{W}]_B^T = \mathbf{0}_{n\times n}$$

Let's restrict ourselves to $\mathbb{R}^3$. Let $[\boldsymbol{v}]_A \in \mathbb{R}^3$ and $[\boldsymbol{v}]_B \in \mathbb{R}^3$ be the coordinate vectors of the physical vector $\boldsymbol{v}$ expressed in coordinate frames A and B, respectively. In this case, $\mathbf{A}_B$ is just the rotation matrix $\mathbf{R}_{A\to B}$ (see Section 2.5).

$$\frac{d[\boldsymbol{v}]_B}{dt} = \mathbf{R}_{A\to B}\left(\frac{d[\boldsymbol{v}]_A}{dt}\right) + [\mathbf{W}]_B\mathbf{R}_{A\to B}[\boldsymbol{v}]_A \tag{3.9}$$

Additionally, in the the three-dimensional case, we have

$$[\mathbf{W}]_B + [\mathbf{W}]_B^T = \mathbf{0}_{3\times 3}$$

Recall from Eq. (1.49) that for an arbitrary 3×3 skew-symmetric matrix $\mathbf{A}$,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 0 & -a_{21} & a_{13} \\ a_{21} & 0 & -a_{32} \\ -a_{13} & a_{32} & 0 \end{bmatrix}$$

In the case of $[\mathbf{W}]_B$, this implies that

$$[\mathbf{W}]_B = \begin{bmatrix} w_{B,11} & w_{B,12} & w_{B,13} \\ w_{B,21} & w_{B,22} & w_{B,23} \\ w_{B,31} & w_{B,32} & w_{B,33} \end{bmatrix} = \begin{bmatrix} 0 & -w_{B,21} & w_{B,13} \\ w_{B,21} & 0 & -w_{B,32} \\ -w_{B,13} & w_{B,32} & 0 \end{bmatrix}$$

Let's define the **angular velocity** of frame B with respect to frame A, *expressed in* frame B, as the three-dimensional coordinate vector

$$[{}^{B/A}\boldsymbol{\omega}]_B \in \mathbb{R}^3$$

In Section 3.4, we will cover the angular velocity vector in-depth. For now, we can ignore its details, beyond the fact that it's simply a three-dimensional vector. Recall from the cross-product section of Section 1.8 that the components of a three-dimensional vector can be arranged into a 3×3 skew-symmetric matrix representing a cross product. In this case,

$$[{}^{B/A}\boldsymbol{\omega}^\times]_B \in \mathbb{R}^{3\times 3}$$

is a skew-symmetric matrix. Since $[\mathbf{W}]_B$ is also a 3×3 skew-symmetric matrix, we define

$$[\mathbf{W}]_B = [{}^{B/A}\boldsymbol{\omega}^\times]_B \tag{3.10}$$

From Eq. (3.5), we know for this case, $[\mathbf{W}]_B$ is defined as

$$[\mathbf{W}]_B = \left(\frac{d\mathbf{R}_{A\to B}}{dt}\right)\mathbf{R}_{A\to B}^T$$

Thus, we have our first formal definition of the angular velocity [3].

$$\boxed{[{}^{B/A}\boldsymbol{\omega}^{\times}]_B = \left(\frac{d\mathbf{R}_{A\to B}}{dt}\right)\mathbf{R}_{A\to B}^T} \tag{3.11}$$

Now, let us return to the main problem at hand, which is finding the time derivative of a coordinate vector expressed in one frame in terms of a coordinate vector in another frame. Substituting Eq. (3.10) into Eq. (3.9),

$$\frac{d[\boldsymbol{v}]_B}{dt} = \mathbf{R}_{A\to B}\left(\frac{d[\boldsymbol{v}]_A}{dt}\right) + [{}^{B/A}\boldsymbol{\omega}^{\times}]_B\mathbf{R}_{A\to B}[\boldsymbol{v}]_A \tag{3.12}$$

Since $\mathbf{R}_{A\to B} \in \mathbb{R}^{3\times 3}$ and $[\boldsymbol{v}]_A \in \mathbb{R}^3$, we know that the product $\mathbf{R}_{A\to B}[\boldsymbol{v}]_A$ is also a three-dimensional vector. Additionally, we know that for an arbitrary column vector $\mathbf{a} \in \mathbb{R}^3$,

$$[{}^{B/A}\boldsymbol{\omega}^{\times}]_B\mathbf{a} = [{}^{B/A}\boldsymbol{\omega}]_B \times \mathbf{a}$$

Substituting this into Eq. (3.12),

$$\boxed{\frac{d[\boldsymbol{v}]_B}{dt} = \mathbf{R}_{A\to B}\left(\frac{d[\boldsymbol{v}]_A}{dt}\right) + [{}^{B/A}\boldsymbol{\omega}]_B \times (\mathbf{R}_{A\to B}[\boldsymbol{v}]_A)} \tag{3.13}$$

3.3.2 Derivative of the Physical Vector

Let $\boldsymbol{v}$ be some physical vector and a physical basis $A = (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$. We can write $\boldsymbol{v}$ as

$$\boldsymbol{v} = \boldsymbol{A}[\boldsymbol{v}]_A = \begin{bmatrix}\hat{\boldsymbol{x}} & \hat{\boldsymbol{y}} & \hat{\boldsymbol{z}}\end{bmatrix}\begin{bmatrix}v_x\\ v_y\\ v_z\end{bmatrix} = v_x\hat{\boldsymbol{x}} + v_y\hat{\boldsymbol{y}} + v_z\hat{\boldsymbol{z}}$$

Differentiating $\boldsymbol{v}$ with respect to time,

$$\begin{aligned}\frac{d\boldsymbol{v}}{dt} &= \frac{d}{dt}\left(v_x\hat{\boldsymbol{x}} + v_y\hat{\boldsymbol{y}} + v_z\hat{\boldsymbol{z}}\right) = \left(\frac{dv_x}{dt}\hat{\boldsymbol{x}} + v_x\frac{d\hat{\boldsymbol{x}}}{dt}\right) + \left(\frac{dv_y}{dt}\hat{\boldsymbol{y}} + v_y\frac{d\hat{\boldsymbol{y}}}{dt}\right) + \left(\frac{dv_z}{dt}\hat{\boldsymbol{z}} + v_z\frac{d\hat{\boldsymbol{z}}}{dt}\right)\\ &= \underbrace{\left(\frac{dv_x}{dt}\hat{\boldsymbol{x}} + \frac{dv_y}{dt}\hat{\boldsymbol{y}} + \frac{dv_z}{dt}\hat{\boldsymbol{z}}\right)}_{\boldsymbol{A}\dfrac{d[\boldsymbol{v}]_A}{dt}} + \underbrace{\left(v_x\frac{d\hat{\boldsymbol{x}}}{dt} + v_y\frac{d\hat{\boldsymbol{y}}}{dt} + v_z\frac{d\hat{\boldsymbol{z}}}{dt}\right)}_{\substack{\text{ambiguous; what do we measure the rate}\\ \text{of change of the basis with respect to?}}}\end{aligned}$$

We can immediately notice that this initial attempt to take the time derivative of a physical vector isn't quite correct. While we do obtain a portion that describes a vector's rate of change with respect to one basis, there is a second portion that is quite ambiguous; it involves rates of change of the *physical* basis vectors, which are abstract quantities. We have no way of quantifying these rates of change unless we define these rates to be ***relative*** to some other reference frame. For example, if we chose this "other" reference frame to be the frame A itself, then the basis vectors of A aren't changing with respect to itself, so we would simply have

$$\frac{d\boldsymbol{v}}{dt} = \boldsymbol{A}\frac{d[\boldsymbol{v}]_A}{dt}$$

However, for any other choice of an "other" reference frame, we'd have

$$\frac{d\boldsymbol{v}}{dt} \neq \boldsymbol{A}\frac{d[\boldsymbol{v}]_A}{dt}$$

Thus, we must clearly distinguish what frame we are taking a temporal derivative with respect to.

We define the **time derivative of a physical vector** *relative to* a coordinate frame B as

$$\boxed{{}^{B}\left(\frac{d\boldsymbol{v}}{dt}\right) = \boldsymbol{B}\frac{d[\boldsymbol{v}]_B}{dt}} \tag{3.14}$$

where $[\boldsymbol{v}]_B$ is the coordinate vector of $\boldsymbol{v}$ expressed in frame B. Now, recall from Eq. (3.19) in Section 3.3.1 that

$$\frac{d[\boldsymbol{v}]_B}{dt} = \mathbf{R}_{A\to B}\left(\frac{d[\boldsymbol{v}]_A}{dt}\right) + [{}^{B/A}\boldsymbol{\omega}]_B \times (\mathbf{R}_{A\to B}[\boldsymbol{v}]_A)$$

Substituting this into Eq. (3.14),

$$\begin{aligned}
{}^{B}\left(\frac{d\boldsymbol{v}}{dt}\right) &= \boldsymbol{B}\left[\mathbf{R}_{A\to B}\left(\frac{d[\boldsymbol{v}]_A}{dt}\right) + [{}^{B/A}\boldsymbol{\omega}]_B \times (\mathbf{R}_{A\to B}[\boldsymbol{v}]_A)\right] \\
&= \underbrace{\boldsymbol{B}\mathbf{R}_{A\to B}}_{\boldsymbol{A}\ \text{(by Eq. (2.20))}}\left(\frac{d[\boldsymbol{v}]_A}{dt}\right) + \boldsymbol{B}\left[[{}^{B/A}\boldsymbol{\omega}]_B \times (\mathbf{R}_{A\to B}[\boldsymbol{v}]_A)\right] \\
&= \underbrace{\boldsymbol{A}\left(\frac{d[\boldsymbol{v}]_A}{dt}\right)}_{\text{apply definition from Eq. (3.14)}} + \boldsymbol{B}\left[[{}^{B/A}\boldsymbol{\omega}]_B \times (\mathbf{R}_{A\to B}[\boldsymbol{v}]_A)\right] \\
&= {}^{A}\left(\frac{d\boldsymbol{v}}{dt}\right) + \boldsymbol{B}\left[[{}^{B/A}\boldsymbol{\omega}]_B \times (\mathbf{R}_{A\to B}[\boldsymbol{v}]_A)\right]
\end{aligned}$$

Since the coordinate vector that results from the operations in the square bracket will be a coordinate vector expressed in frame B, and since that resultant coordinate vector is multiplied by the frame matrix $\boldsymbol{B}$, we can apply the concepts from Section 2.6.1 and say

$$\boxed{{}^{B}\left(\frac{d\boldsymbol{v}}{dt}\right) = {}^{A}\left(\frac{d\boldsymbol{v}}{dt}\right) + {}^{B/A}\boldsymbol{\omega} \times \boldsymbol{v}} \tag{3.15}$$

3.3.3 Rate Vectors

In Section 3.3.2, we showed that the time derivative of a physical vector relative to coordinate frame B can be written in terms of its time derivative relative to coordinate frame A as

$${}^{B}\left(\frac{d\boldsymbol{v}}{dt}\right) = {}^{A}\left(\frac{d\boldsymbol{v}}{dt}\right) + {}^{B/A}\boldsymbol{\omega} \times \boldsymbol{v}$$

However, we completely glossed over the fact that $\boldsymbol{v}$ may *itself* be a vector representing a time derivative of another physical vector relative to some coordinate frame.

Let $\boldsymbol{v}$ be a physical vector that is defined as the time derivative of another physical vector $\boldsymbol{x}$. Then $\boldsymbol{v}$ is a **rate vector**, and we should denote which frame the rate of change of $\boldsymbol{x}$ is measured relative to.

Convention 19: Notation for rate vectors.

If $\boldsymbol{v}$ is a rate vector that defines the time rate of change of another vector relative to coordinate frame A, then we denote it as

$${}^{A}\boldsymbol{v}$$

Convention 20: Notation for time derivatives of rate vectors.

Consider the rate vector ${}^{A}\boldsymbol{v}$ measured relative to coordinate frame A. The time derivative of ${}^{A}\boldsymbol{v}$ relative to frame A is denoted

$$ {}^{A}\left(\frac{d\boldsymbol{v}}{dt}\right) = {}^{A}\left[\frac{d({}^{A}\boldsymbol{v})}{dt}\right] $$

Using these conventions, we can write Eq. (3.15) more rigorously as

$$ \boxed{ {}^{B}\left(\frac{d\boldsymbol{v}}{dt}\right) = \underbrace{{}^{A}\left(\frac{d\boldsymbol{v}}{dt}\right)}_{\substack{\text{contribution from}\\ \text{rate of change}\\ \text{w.r.t. frame } A}} + \underbrace{{}^{B/A}\boldsymbol{\omega} \times {}^{A}\boldsymbol{v}}_{\substack{\text{contribution from}\\ \text{rotation of frame } B\\ \text{relative to frame } A}} } \tag{3.16} $$

3.4 Angular Velocity

3.5 The Golden Rule of Kinematics

In Sections 3.2 and 3.3, we derived two equations; one that is fundamental to the study of translation kinematics, and another that is fundamental to its simulation. There is no "standard" name for these equations; some names include the *transport theorem*, *transport equation*, *rate of change transport theorem*, *basic kinematic equation*, and *Euler derivative transformation formula* [53]. In this text, we refer to these equations as the ***Golden Rule*** to signify their importance to study of motion[2].

Consider two coordinate frames, $A = (\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$ and $B = (\hat{\mathbf{a}}, \hat{\mathbf{b}}, \hat{\mathbf{c}})$, with the same origin O, where A is rotating with respect to B with angular velocity ${}^{A/B}\boldsymbol{\omega}$. Consider an arbitrary vector, $\boldsymbol{v}$. If we know $\boldsymbol{v}$ and its derivative relative to frame B, then we can find its derivative relative to frame A as

$$ \boxed{ {}^{B}\left(\frac{d\boldsymbol{v}}{dt}\right) = {}^{A}\left(\frac{d\boldsymbol{v}}{dt}\right) + {}^{B/A}\boldsymbol{\omega} \times {}^{A}\boldsymbol{v} } \tag{3.17} $$

Note that this equation was derived in Section 3.3.

We can use Eq. (3.17) to develop equations that describe the motion and vector properties of physical bodies. However, to simulate kinematics, we need the same equation but in a coordinate vector form. TODO

$$ \boxed{ {}^{B}\left(\frac{d[\boldsymbol{v}]_B}{dt}\right) = (\mathbf{R}_{A\to B}) \, {}^{A}\left(\frac{d[\boldsymbol{v}]_A}{dt}\right) + \left({}^{B/A}\boldsymbol{\omega}_B\right) \times \left[(\mathbf{R}_{A\to B})\left({}^{A}\mathbf{v}_A\right)\right] } \tag{3.18} $$

$$ \boxed{ \frac{d[\boldsymbol{v}]_B}{dt} = \mathbf{R}_{A\to B}\left(\frac{d[\boldsymbol{v}]_A}{dt}\right) + [{}^{B/A}\boldsymbol{\omega}]_B \times (\mathbf{R}_{A\to B}[\boldsymbol{v}]_A) } \tag{3.19} $$

[2] This name is used in Stanford's classical dynamics course (AA 242A).

3.6 Derivative of the Rotation Matrix

In Section 3.3.1, we derived Eq. (3.11), repeated here.

$$\left[{}^{B/A}\boldsymbol{\omega}_B\right]_\times = \left(\frac{d\mathbf{R}_{A\to B}}{dt}\right)\mathbf{R}_{A\to B}^T \tag{3.11}$$

Multiplying both sides by $\mathbf{R}_{A\to B}$, and noting that $\mathbf{R}_{A\to B}^T\mathbf{R}_{A\to B} = \mathbf{I}_{3\times 3}$ [58],

$$\boxed{\frac{d\mathbf{R}_{A\to B}}{dt} = \left[{}^{B/A}\boldsymbol{\omega}_B\right]_\times \mathbf{R}_{A\to B}} \tag{3.20}$$

TODO: algorithm

4

Attitude Kinematics

4.1 Body Frame and World Frame

In aerospace, the **body frame** of a vehicle is a coordinate frame with its origin at the center of mass of the vehicle, and with axes aligned with the major geometric features of the vehicle. In general, the 1st axis of the body frame (x_b) aligns with the **longitudinal** (length-wise) axis of the vehicle. The third axis of the body frame (z_b) is typically chosen such that $x_b z_b$ is a plane of symmetry of the vehicle (since aerospace vehicles are typically symmetrical). The second axis of the body frame (y_b) then completes the right-handed triad (for airplanes, the y_b axis points approximately along a wing). The body frames of a rocket and an airplane are displayed in Figs. 4.1 and 4.2, respectively, below. Note that the label "cm" denotes the center of mass of the vehicle.

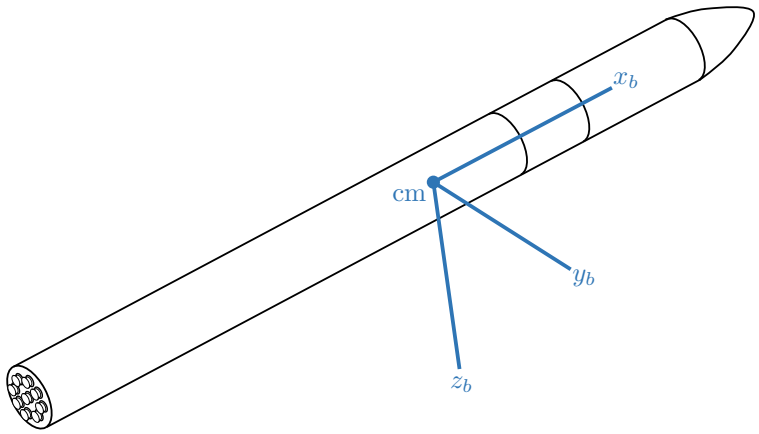


Figure 4.1: Rocket body frame.

Loosely speaking, the **attitude** of a vehicle is its orientation with respect to its environment. More rigorously, it is defined as the orientation of a vehicle's body frame with respect to the **world frame**. The world frame is essentially any other coordinate frame that we use to define a vehicle's attitude with respect to. The world and body frame are illustrated in Fig. 4.3.

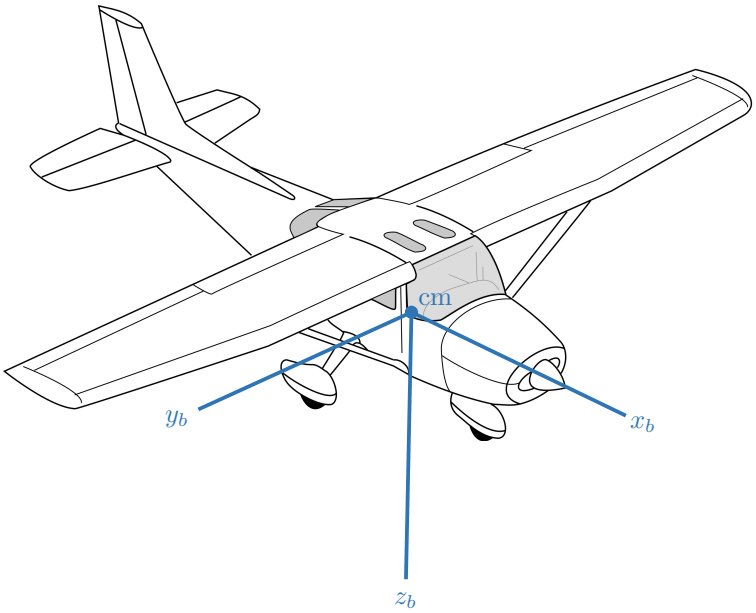


Figure 4.2: Airplane body frame.

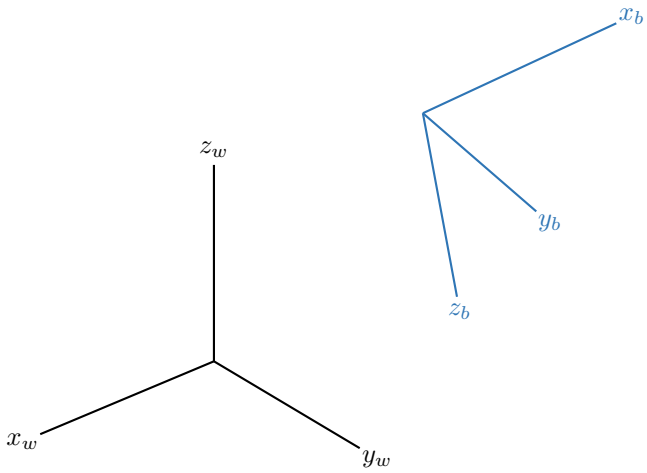


Figure 4.3: World and body frames.

4.2 Attitude, Rotation, and Direction Cosine Matrices

To simulate both the translational and attitude dynamics of aerospace vehicles, we need a representation of its **attitude**, or orientation, with respect to its environment.

To motivate the initial development of an attitude parameterization, recall the world and body frames introduced in the previous section. A vehicle's rotational motion is most convenient to describe in the body frame, but it is (in the most general case) coupled to its translational motion, which is described in a world frame. Furthermore, many forces are also originally expressed in a world frame, so we have to transform them so we can apply them to the vehicle in the body frame. Let's consider an arbitrary force, $\boldsymbol{F}$, resolved in the world frame.

$$[\boldsymbol{F}]_w = \begin{bmatrix} F_{x_w} \\ F_{y_w} \\ F_{z_w} \end{bmatrix}$$

To express the force in body frame, we just need to apply the rotation matrix $\mathbf{R}_{\text{world}\to\text{body}}$.

$$[\boldsymbol{F}]_{\text{body}} = \mathbf{R}_{\text{world}\to\text{body}}[\mathbf{F}]_{\text{world}}$$

Immediately, we note that $\mathbf{R}_{\text{world}\to\text{body}}$ encodes information regarding the attitude of the vehicle.

Going into more detail, the world frame has basis $(\hat{\boldsymbol{x}}_w, \hat{\boldsymbol{y}}_w, \hat{\boldsymbol{z}}_w)$ while the body frame is defined by the basis $(\hat{\boldsymbol{x}}_b, \hat{\boldsymbol{y}}_b, \hat{\boldsymbol{z}}_b)$. At the beginning of a simulation, we will always know the basis vectors of the body frame expressed in the world frame; these serve as an initial condition for the simulation.

$$[\hat{\boldsymbol{x}}_b]_{\text{world}}, \quad [\hat{\mathbf{y}}_b]_{\text{world}}, \quad [\hat{\boldsymbol{z}}_b]_{\text{world}}$$

Recall from Section 2.5.2 that we can construct the rotation matrix between two frames by taking the basis vectors of one frame and expressing them in the other frame. In this case, Eq. (2.17) gives us

$$\mathbf{R}_{\text{world}\to\text{body}} = \begin{pmatrix} [\hat{\boldsymbol{x}}_b]^T_{\text{world}} \\ [\hat{\boldsymbol{y}}_b]^T_{\text{world}} \\ [\hat{\boldsymbol{z}}_b]^T_{\text{world}} \end{pmatrix}$$

As the simulation progresses, we essentially keep track of $\mathbf{R}_{\text{world}\to\text{body}}$ (usually in the form of some other attitude parameterization that can be converted to/from this form).

The **attitude matrix**, $\mathbf{A}$, is the coordinate transformation that maps vectors in the world frame to the body frame [56, p. 411]. More specifically, to match the language used in this text, the attitude matrix is the rotation matrix that takes vectors expressed in the world frame and resolves them in the body frame.

$$\mathbf{A} = \mathbf{R}_{\text{world}\to\text{body}}$$

In most practical cases, we can have multiple different world frames and multiple different body-fixed frames. Therefore, it is imperative that we also subscript the attitude matrix to specifically denote which coordinate transformation it is defining.

$$\boxed{\mathbf{A}_{\text{world}\to\text{body}} = \mathbf{R}_{\text{world}\to\text{body}}} \tag{4.1}$$

Note that the attitude matrix is also commonly referred to as the **direction cosine matrix (DCM)**, since it can be constructed using the direction cosines (see Section **??**) between each combination of basis vectors between the two frames. However, constructing an attitude matrix using direction cosines is inefficient and not used in practice, so we completely omit this definition here to avoid confusion.

4.2.1 Attitude vs. Rotation Parameterization

Through Eq. (4.1), we have just shown that the parameterization of attitude is essentially just a parameterization of rotation. Thus, we broaden the scope of the remaining sections of this chapter and discuss these parameterizations more generally as paramaterizations of *rotation*, and not just of attitude.

4.2.2 Disadvantages

Recall from Section 2.5.1 that we can chain rotation matrices as

$$\mathbf{R}_{A\to C} = \mathbf{R}_{B\to C}\mathbf{R}_{A\to B}$$

Generally, even if each individual rotation matrix is an orthogonal matrix, the product of multiple rotation matrices starts getting less and less orthogonal due to floating point errors. Due to this potential loss of orthogonality, rotation matrices can be disadvantageous to use, especially when dealing with successive rotations (recall from Section 1.6 that rotation matrices must be orthogonal) [18]. While we can convert the resultant rotation matrix to its nearest orthonormal representation, it is more difficult and inefficient than renormalizations needed for other rotation parameterizations (for an example, look at the quaternion normalization in Section 4.5.2).

Additionally, while we *can* easily obtain the time derivative of the rotation matrix (see Section 3.6) and thus use it as a state variable when simulating an aerospace vehicle, it would (a) make the equations of attitude motion much more complicated and (b) require **nine** equations just to describe the attitude rate of change (one for each element of the rotation matrix). Furthermore, solving the differential equations numerically would almost certainly yield rotation matrices that are not orthogonal.

4.3 Euler Angles: Yaw, Pitch, and Roll

In the previous section, we defined the attitude of a vehicle using using the attitude matrix, $\mathbf{A}_{\text{world}\to\text{body}}$, which was defined as being equivalent to the rotation matrix $\mathbf{R}_{\text{world}\to\text{body}}$. Recall the 3-2-1 rotation sequence defined in Section 1.6.4. *Any* 3D rotation can be described in terms of the 3-2-1 rotation sequence.

The **3-2-1 Euler angles** are the angles defining the 3-2-1 rotation sequence. In aerospace, we use a 3-2-1 rotation sequence to transform vectors from the world frame to the body frame. In this specific context, we refer to this rotation sequence as the **yaw-pitch-roll** sequence. As the name implies, we first perform the yaw rotation, then the pitch rotation, and finally the roll rotation. The rotation sequence can be outlined in a more detailed fashion as:

1. **1st rotation (yaw rotation):** rotation about the world z_w-axis by the **yaw angle**, ψ. This rotation transforms a vector from the $x_w y_w z_w$ coordinate system (world frame) to the intermediate coordinate system, $x'y'z'$, where $z' = z$.
2. **2nd rotation (pitch rotation):** rotation about the y'-axis by the **pitch angle**, θ. This rotation transforms a vector from the $x'y'z'$ coordinate system to another intermediate coordinate system, $x''y''z''$, where $y'' = y'$.
3. **3rd rotation (roll rotation):** rotation about the x''-axis by the **roll angle**, ϕ. This rotation transforms a vector from the $x''y''z''$ coordinate system to the $x_b y_b z_b$ coordinate system (body frame), where $x_b = x''$.

$$\boxed{\begin{array}{l}\psi = \text{yaw angle} = \text{angle of 1st applied rotation, performed about } z_w \text{ (3rd) axis} \\ \theta = \text{pitch angle} = \text{angle of 2nd applied rotation, performed about } y' \text{ (2nd) axis} \\ \phi = \text{roll angle} = \text{angle of 3rd applied rotation, performed about } x_b = x'' \text{ (1st) axis}\end{array}} \tag{4.2}$$

In practice, the world frame and body frame have specific names and definitions. Like we labelled the attitude matrix, we can also label the angles with the specific transformation they represent. For example:

1. $\psi_{\text{world}\to\text{body}}$ = yaw angle (world to body rotation)
2. $\theta_{\text{world}\to\text{body}}$ = pitch angle (world to body rotation)
3. $\phi_{\text{world}\to\text{body}}$ = roll angle (world to body rotation)

4.3.1 Yaw Angle

The domain of the yaw angle, ψ, is [48, p. 12]

$$\boxed{-\pi < \psi \leq \pi} \tag{4.3}$$

A visual depiction of the yaw angle is shown in Fig. 4.4. This simplified depiction assumes that the body z_b-axis is aligned with the world z_w-axis.

For an airplane in standard flight, a positive yaw angle corresponds to a "nose-right" attitude. *This is because the z_w and z_b axes are pointing "into the page", and yaw is positive when it defines a counterclockwise rotation about the z_w-axis.*

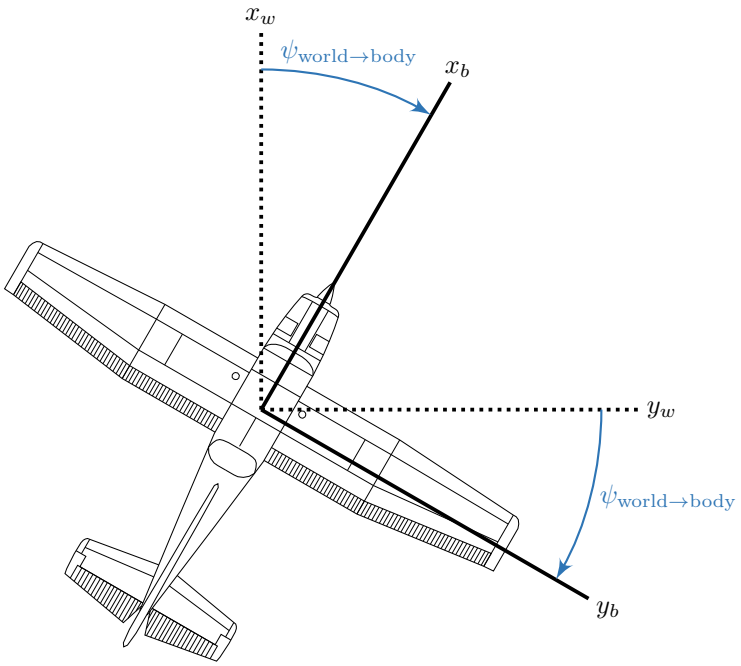


Figure 4.4: Yaw angle.

4.3.2 Pitch Angle

The domain of the pitch angle, θ, is [48, p. 12]

$$\boxed{-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}} \tag{4.4}$$

A visual depiction of the pitch angle is shown in Fig. 4.5. This simplified depiction assumes that the body y_b-axis is aligned with the world y_w-axis.

> For an airplane in standard flight, a positive pitch angle corresponds to a "nose-up" attitude. *This is because the y_w and y_b axes are pointing "into the page", and pitch is positive when it defines a counterclockwise rotation about the y_w-axis.*

4.3.3 Roll Angle

The domain of the roll angle, ϕ, is [48, p. 12]

$$\boxed{-\pi < \phi \leq \pi} \tag{4.5}$$

A visual depiction of the roll angle is shown in Fig. 4.6. This simplified depiction assumes that the body x_b-axis is aligned with the world x_w-axis.

> For an airplane in standard flight, a positive roll angle corresponds to a "bank-right" attitude. *This is because the x_w and x_b axes are pointing "out of the page", and roll is positive when it defines a counterclockwise*

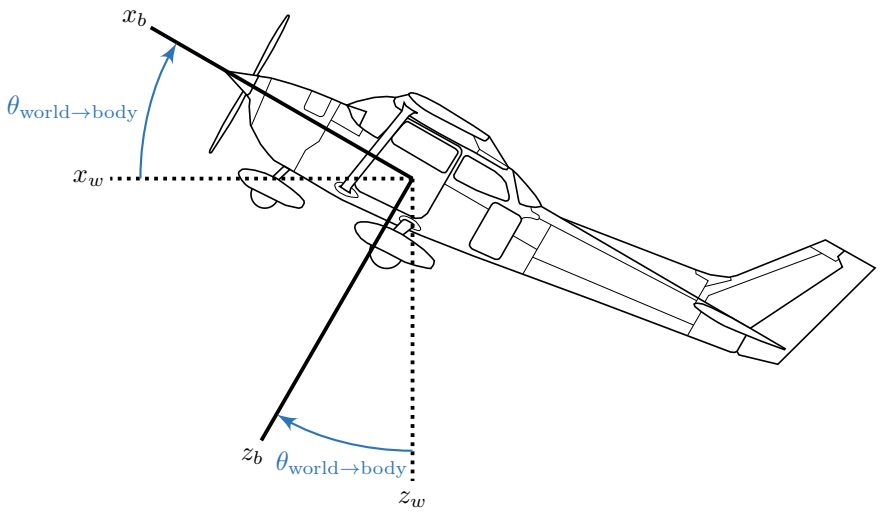


Figure 4.5: Pitch angle.

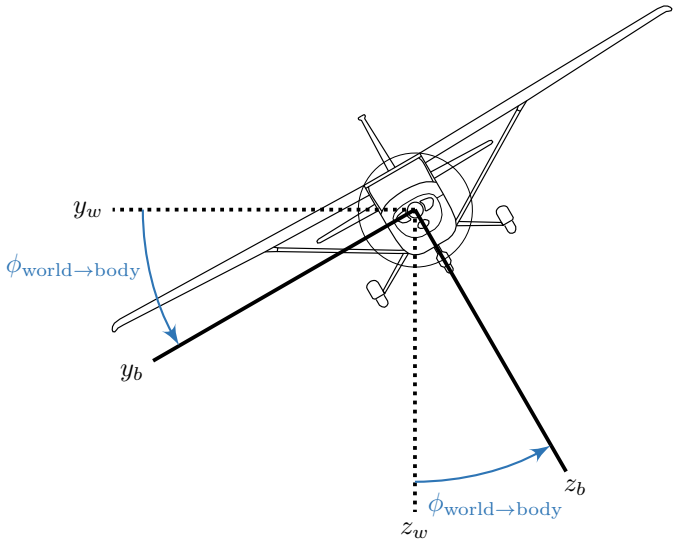


Figure 4.6: Roll angle.

rotation about the x_w-axis.

4.3.4 Conversions Between Rotation Matrices and Euler Angles

> Since we can have a rotation matrix between any two arbitrary coordinate frames, we drop the "world $\rightarrow$ body"subscripts here. However, note that in practice, it is imperative that we label the angles; otherwise, there is a complete ambiguity with regards to what transformation the angles represent.

Recall Eq. (1.44), repeated below, defining the rotation matrix for the 3-2-1 rotation sequence.

$$\mathbf{R}_{321}(\theta_1, \theta_2, \theta_3) = \begin{bmatrix} \cos\theta_2 \cos\theta_1 & \cos\theta_2 \sin\theta_1 & -\sin\theta_2 \\ -\cos\theta_3 \sin\theta_1 + \sin\theta_3 \sin\theta_2 \cos\theta_1 & \cos\theta_3 \cos\theta_1 + \sin\theta_3 \sin\theta_2 \sin\theta_1 & \sin\theta_3 \cos\theta_2 \\ \sin\theta_3 \sin\theta_1 + \cos\theta_3 \sin\theta_2 \cos\theta_1 & -\sin\theta_3 \cos\theta_1 + \cos\theta_3 \sin\theta_2 \sin\theta_1 & \cos\theta_3 \cos\theta_2 \end{bmatrix} \tag{1.44}$$

Since the yaw-pitch-roll sequence is a 3-2-1 sequence, the rotation matrix can be written in terms of the yaw, pitch, and roll angles as

$$\mathbf{R} = \mathbf{R}_{321}(\psi, \theta, \phi) \tag{4.6}$$

From Eqs. (1.44) and 4.6, we get

$$\boxed{\mathbf{R} = \begin{bmatrix} \cos\theta \cos\psi & \cos\theta \sin\psi & -\sin\theta \\ -\cos\phi \sin\psi + \sin\phi \sin\theta \cos\psi & \cos\phi \cos\psi + \sin\phi \sin\theta \sin\psi & \sin\phi \cos\theta \\ \sin\phi \sin\psi + \cos\phi \sin\theta \cos\psi & -\sin\phi \cos\psi + \cos\phi \sin\theta \sin\psi & \cos\phi \cos\theta \end{bmatrix}} \tag{4.7}$$

However, since we already have defined Algorithm 4 for computing $\mathbf{R}_{321}(\theta_1, \theta_2, \theta_3)$, we can just wrap around that algorithm to obtain the rotation matrix from the 3-2-1 Euler angles.

Algorithm 10: `eul2mat_321`

3-2-1 Euler angles (yaw, pitch, and roll) to rotation matrix.

Inputs:

- $\psi \in \mathbb{R}$ - yaw angle (1st rotation, about 3rd axis) [rad]
- $\theta \in \mathbb{R}$ - pitch angle (2nd rotation, about 2nd axis) [rad]
- $\phi \in \mathbb{R}$ - roll angle (3rd rotation, about 1st axis) [rad]

Procedure:

$\mathbf{R} = \texttt{rot321}(\psi, \theta, \phi)$ (Algorithm 4)

return $\mathbf{R}$

Outputs:

- $\mathbf{R} \in \mathbb{R}^{3\times3}$ - rotation matrix for the 3-2-1 (yaw-pitch-roll) rotation sequence

Test Cases:

- See Appendix A.1.1.

We can also perform the inverse conversion; given a rotation matrix, we can obtain the yaw-pitch-roll rotation sequence that would generate that rotation matrix. Dividing R_{12} by R_{11},

$$\frac{R_{12}}{R_{11}} = \frac{\cos\theta \sin\psi}{\cos\theta \cos\psi} = \frac{\sin\psi}{\cos\psi} = \quad \rightarrow \quad \tan\psi = \frac{R_{12}}{R_{11}}$$

Since the domain of ψ spans all four quadrants, we need to solve for ψ in the correct quadrant; to do this, we can use the 4-quadrant inverse tangent.

$$\psi = \operatorname{arctan2}(R_{12}, R_{11}) \tag{4.8}$$

Since the domain of ψ is $(-\pi, \pi]$, and since the range of the four-quadrant inverse tangent function is $[-\pi, \pi]$, we don't have to perform any additional quadrant checks.

Next, from R_{13},

$$R_{13} = -\sin\theta \quad \rightarrow \quad \sin\theta = -R_{13} \quad \rightarrow \quad \theta = \arcsin(-R_{13})$$

$$\theta = -\arcsin R_{13}$$

Since the domain of θ is $[-\pi/2, \pi/2]$, and since the range of the inverse sine function is $[-\pi/2, \pi/2]$, we don't have to perform any additional quadrant checks. However, in practice, it is common for $|R_{13}|$ to exceed 1 due to finite precision computer arithmetic, which is an issue since the domain of the inverse sine function is $[-1, 1]$. We can protect against this by including a check on $|A|_{13}$ [48, p. 13].

$$\theta = \begin{cases} -\arcsin R_{13}, & |R_{13}| < 1 \\ -\left(\dfrac{\pi}{2}\right) \operatorname{sgn} R_{13}, & |R_{13}| \geq 1 \end{cases} \tag{4.9}$$

Finally, dividing R_{23} by R_{33},

$$\frac{R_{23}}{R_{33}} = \frac{\sin\phi\cos\theta}{\cos\phi\cos\theta} = \frac{\sin\phi}{\cos\phi} \quad \rightarrow \quad \tan\phi = \frac{R_{23}}{R_{33}}$$

Since the domain of ϕ spans all four quadrants, we need to solve for ϕ in the correct quadrant; to do this, we can use the 4-quadrant inverse tangent.

$$\phi = \operatorname{arctan2}(R_{23}, R_{33}) \tag{4.10}$$

Since the domain of ϕ is $(-\pi, \pi]$, and since the range of the four-quadrant inverse tangent function is $[-\pi, \pi]$, we don't have to perform any additional quadrant checks [17], [48, p. 12].

Singularity: Pitch-Up Case

However, there remains a critical issue; the 3-2-1 Euler angles have a **singularity** when

$$\theta = \pm\frac{\pi}{2}$$

We refer to the case where $\theta = \pi/2$ as the **pitch-up case**. At this condition, we also have $R_{13} = -1$. As mentioned before, R_{13} can be ever so slightly less than -1 due to finite precision arithmetic, so we define

$$\boxed{R_{13} \leq -1 \quad \rightarrow \quad \theta = \frac{\pi}{2} \quad \rightarrow \quad \text{pitch-up case}} \tag{4.11}$$

For the pitch-up case, the rotation matrix given by Eq. (4.7) becomes

$$\begin{aligned} \mathbf{R} &= \begin{bmatrix} 0 & 0 & -1 \\ -\cos\phi\sin\psi + \sin\phi\cos\psi & \cos\phi\cos\psi + \sin\phi\sin\psi & 0 \\ \sin\phi\sin\psi + \cos\phi\cos\psi & -\sin\phi\cos\psi + \cos\phi\sin\psi & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & -1 \\ \sin\phi\cos\psi - \cos\phi\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\psi & 0 \\ \cos\phi\cos\psi + \sin\phi\sin\psi & -(\sin\phi\cos\psi - \cos\phi\sin\psi) & 0 \end{bmatrix} \end{aligned}$$

Trigonometric identities give us

$$\sin(\phi - \psi) = \sin\phi\cos\psi - \cos\phi\sin\psi$$
$$\cos(\phi - \psi) = \cos\phi\cos\psi + \sin\phi\sin\psi$$

The rotation matrix can then be simplified to

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & -1 \\ \sin(\phi - \psi) & \cos(\phi - \psi) & 0 \\ \cos(\phi - \psi) & -\sin(\phi - \psi) & 0 \end{bmatrix}$$

If we define the auxiliary angle

$$\alpha = \phi - \psi$$

then the rotation matrix is simply

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & -1 \\ \sin\alpha & \cos\alpha & 0 \\ \cos\alpha & -\sin\alpha & 0 \end{bmatrix}$$

This implies that there are an infinite number of combinations of ψ and ϕ that will recover this same rotation matrix. We choose

$$\psi = 0 \tag{4.12}$$

Convention 21: Roll and yaw angles for the 3-2-1 Euler angle singularities.

When we encounter the $\theta = \pm\pi/2$ singularities while converting from another rotation parameterization to the 3-2-1 Euler angles, we manually set the yaw angle to be

$$\psi = 0$$

and then compute the roll angle, ϕ, accordingly.

The rotation matrix then becomes

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & -1 \\ \sin\phi & \cos\phi & 0 \\ \cos\phi & -\sin\phi & 0 \end{bmatrix}$$

From this rotation matrix, we can extract ϕ in a similar way as before.

$$\tan\phi = \frac{\sin\phi}{\cos\phi} = \frac{R_{21}}{R_{31}} \quad \rightarrow \quad \phi = \text{arctan2}(R_{21}, R_{31}) \tag{4.13}$$

Note that

Singularity: Pitch-Down Case

Now, let's handle the singularity when $\theta = -\pi/2$. We refer to the case where $\theta = -\pi/2$ as the **pitch-down case**. At this condition, we also have $R_{13} = 1$. As mentioned before, R_{13} can be ever so slightly greater than 1 due to finite precision arithmetic, so we define

$$\boxed{R_{13} \geq 1 \quad \rightarrow \quad \theta = -\frac{\pi}{2} \quad \rightarrow \quad \text{pitch-down case}} \tag{4.14}$$

For the pitch-down case, the rotation matrix given by Eq. (4.7) becomes

$$\mathbf{R} \begin{bmatrix} 0 & 0 & 1 \\ -\cos\phi\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\psi - \sin\phi\sin\psi & 0 \\ \sin\phi\sin\psi - \cos\phi\cos\psi & -\sin\phi\cos\psi - \cos\phi\sin\psi & 0 \end{bmatrix}$$

Once again, we let

$$\psi = 0 \tag{4.15}$$

resulting in

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & 1 \\ -\sin\phi & \cos\phi & 0 \\ -\cos\phi & -\sin\phi & 0 \end{bmatrix}$$

Note that $\sin(-\phi) = -\sin\phi$ and $\cos(-\phi) = \cos\phi$. Then we have

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & 1 \\ \sin(-\phi) & \cos(-\phi) & 0 \\ -\cos(-\phi) & \sin(-\phi) & 0 \end{bmatrix}$$

From this rotation matrix, we can extract ϕ in a similar way as before.

$$\tan(-\phi) = \frac{\sin(-\phi)}{\cos(-\phi)} = \frac{R_{32}}{R22} \quad \rightarrow \quad -\phi = \arctan2(R_{32}, R_{22}) \quad \rightarrow \quad \phi = -\arctan2(R_{32}, R_{22}) \tag{4.16}$$

Final Set of Equations

From Eqs. (4.8), (4.12), and (4.15), we can arrive at Eq. (4.17) defining the yaw angle. The pitch angle is still defined by Eq. (4.9), which we repeat as Eq. (4.18) below. From Eqs. (4.10), (4.13), and (4.16), we can arrive at Eq. (4.19) defining the roll angle.

$$\boxed{\psi = \begin{cases} \arctan2(R_{12}, R_{11}), & |R_{13}| < 1 \\ 0, & |R_{13}| \geq 1 \end{cases}} \tag{4.17}$$

$$\boxed{\theta = \begin{cases} -\arcsin R_{13}, & |R_{13}| < 1 \\ -\left(\frac{\pi}{2}\right) \operatorname{sgn} R_{13}, & |R_{13}| \geq 1 \end{cases}} \tag{4.18}$$

$$\boxed{\phi = \begin{cases} \arctan2(R_{23}, R_{33}), & |R_{13}| < 1 \\ \arctan2(R_{21}, R_{31}), & R_{13} \leq -1 \\ -\arctan2(R_{32}, R_{22}), & R_{13} \geq 1 \end{cases}} \tag{4.19}$$

Algorithm

Algorithm 11: `mat2eul_321`

Rotation matrix to 3-2-1 Euler angles (yaw, pitch, and roll).

Inputs:

- $\mathbf{R} \in \mathbb{R}^{3\times3}$ - rotation matrix for the 3-2-1 (yaw-pitch-roll) rotation sequence

Procedure:

1. Booleans that store whether or not we are in a singular case, and if so, which one.

   `pitch_up` $= (R_{13} \leq -1)$
   `pitch_down` $= (R_{13} \geq 1)$
   `singular` $=$ (`pitch_up` **or** `pitch_down`)

2. Yaw angle [rad].

if `singular`
$\quad \psi = 0$
else
$\quad \psi = \operatorname{arctan2}(R_{12}, R_{11})$
end

3. Pitch angle $[\mathrm{rad}]$.

if `singular`
$\quad \theta = -\left(\frac{\pi}{2}\right) \operatorname{sgn} R_{13}$
else
$\quad \theta = -\arcsin R_{13}$
end

4. Roll angle $[\mathrm{rad}]$.

if `pitch_up`
$\quad \phi = \operatorname{arctan2}(R_{21}, R_{31})$
else if `pitch_down`
$\quad \phi = -\operatorname{arctan2}(R_{32}, R_{22})$
else
$\quad \phi = \operatorname{arctan2}(R_{23}, R_{33})$
end

5. Return the results.

return ψ, θ, ϕ

Outputs:

- $\psi \in \mathbb{R}$ - yaw angle (1st rotation, about 3rd axis) $[\mathrm{rad}]$
- $\theta \in \mathbb{R}$ - pitch angle (2nd rotation, about 2nd axis) $[\mathrm{rad}]$
- $\phi \in \mathbb{R}$ - roll angle (3rd rotation, about 1st axis) $[\mathrm{rad}]$

Note:

- $\psi \in (-\pi, \pi]$
- $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$
- $\phi \in (-\pi, \pi]$
- When a singularity corresponding to the pitch angles of $\theta = \pm\pi/2$ is encountered, the yaw angle (ψ) is set to 0 and the roll angle (ϕ) is determined accordingly.

Test Cases:

- See Appendix A.1.2.

4.3.5 Gimbal Lock

Recall that in Section 4.3.4, we observed that the 3-2-1 Euler angles have singularities at $\theta = \pm\pi/2$. This singularity implied that when $\theta = \pm\pi/2$, any applied roll angle would only be seen as a rotation in yaw.

At the singularities at $\theta = \pm\pi/2$, any combination of roll and yaw rotations will result only in a rotation in yaw. Thus, we say can say that any rotation not involving pitch is *locked* to be a rotation in yaw. If we recall the pitch angle diagram in Fig. 4.5, we can note that $\pi/2$ would result in the plane would be in a vertical attitude with respect to the world frame. If we tried applying a roll to the visualization, the visualization wouldn't roll, but rather yaw. Unless we modified pitch, there would be no way to make the visualization of the plane roll if we simply constructed an rotation matrix using the Euler angles.

> For aircraft, the world frame typically has its $x_w y_w$ plane oriented parallel to the Earth's surface. For passenger planes, this singularity at $\theta = \pi/2$ does not pose much of a problem, since these planes will never fly near pitch angles of $\theta = \pi/2$. However, fighter jets and aerobatic aircraft will more often experience conditions near $\theta = \pi/2$, and rockets will almost always have a pitch angle at or near $\pi/2$ at some point in their launch trajectory.

If we construct an rotation matrix with any ψ and ϕ but with $\theta = \pi/2$, we cannot (generally[1]) recover those same values of ψ and ϕ from that same rotation matrix. As a numerical example, consider $\psi = \pi/4$, $\theta = \pi/2$, and $\phi = \pi/3$. If we construct the rotation matrix with Algorithm 10, we get

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & -1 \\ 0.2588 & 0.9659 & 0 \\ 0.9659 & -0.2588 & 0.0000 \end{bmatrix}$$

If we try recovering ϕ, θ, and ψ using Algorithm 11, we get

$$\begin{aligned} \psi &= 0 \\ \theta &= \frac{\pi}{2} \\ \phi &= -0.2618 \end{aligned}$$

which is not what we started out with.

Additionally, near the singularities at $\theta = \pm\pi/2$, the roll and yaw angles begin varying rapidly, and are changing infinitely fast as the attitude moves through a pitch of $\theta = \pm\pi/2$. This corresponds to extremely high Euler angle rates, which can lead to unphysical results when integrating the equations of motion using an Euler angle attitude parameterization.

In the physical world, this phenomena manifests itself as **gimbal lock**. Historically, 3-axis gyroscopes were often used for attitude determination on spacecraft. However, when any two of the three gimbals of a gyroscope are aligned, those two gimbals become locked, and the attitude with respect to the axis of those locked gimbals cannot be determined. Additionally, as the attitude moves through the singularity, the roll and yaw angles begin moving infinitely fast, to the point where they exceed the mechanical capabilities of the gimbals. In practice, it is usually the latter case that actually causes the gimbals to lock. For example, on Apollo 11, as the vehicle's attitude approached a pitch of $90°$, there was logic in place to flip one of the gimbals by $180°$ (using a torque motor) as the singularity was neared to avoid gimbal lock. However, the commanded torque was beyond the capabilities of the motor, so the system gave up and froze the gimbals instead. This required the astronauts to adjust the pitch of the spacecraft to move it away from the singularity, and then manually realign the gyroscope. In the physical case, gimbal lock can be avoided by adding a fourth gimbal to provide redundancy. This fourth gimbal is actively driven to maintain a large angle with respect to the roll and yaw axes [18].

> These concept of gimbal lock is hard to grasp (and even harder to explain). Therefore, we suggest the following additional resources:
> - http://webserver2.tecgraf.puc-rio.br/~mgattass/demo/EulerAnglesRotations-GimballLock/euler.html

[1] Algorithm 10 defaults to returning $\psi = 0$ if a singularity is encountered, in which case the roll angle could be correctly recovered.

- https://www.youtube.com/watch?v=zc8b2Jo7mno&t=1s
- https://en.wikipedia.org/wiki/Gimbal_lock
- https://matthew-brett.github.io/transforms3d/gimbal_lock.html

4.4 Axis-Angle Representation

Any 3D rotation can be viewed as a single roll about a single axis. The axis of rotation (which we refer to as the **Euler axis**) is defined by a vector, $\mathbf{e} \in \mathbb{R}^3$, called the **principal rotation vector**. The angle of rotation, Φ, is referred to as the **principal angle** [52, pp. 16–19], [33]. For any 3D rotation, there are two possible axis-angle representations:

1. rotation of Φ about $\mathbf{e}$
2. rotation of $-\Phi$ about $-\mathbf{e}$

To maintain a consistent sign convention, we restrict Φ to be in the interval $[0, \pi]$, and then adjust the sign on $\mathbf{e}$ accordingly.

Convention 22: Interval of the principal angle.

The interval of the principal angle is

$$\Phi \in [0, \pi]$$

Additionally, we assume that the principal rotation vector is a unit vector.

Convention 23: Principal rotation vector magnitude.

Since it is a unit vector, the principal rotation vector has a magnitude of 1.

$$\|\mathbf{e}\| = \sqrt{e_1^2 + e_2^2 + e_3^2} = 1$$

To protect against numerical errors induced by floating point precision, we always normalize the principal rotation vector using

$$\mathbf{e} = \frac{\mathbf{e}}{\|\mathbf{e}\|}$$

when converting from another rotation parameterization to the axis-angle representation.

4.4.1 Relationship With Rotation Matrices

Consider the rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$. Our goal is to find the principal rotation vector, $\mathbf{e} \in \mathbb{R}^3$, and the principal angle, $\Phi \in \mathbb{R}$, corresponding to the rotation matrix $\mathbf{R}$.

The principal angle, Φ, is related to the elements of the rotation matrix as

$$\boxed{\cos\Phi = \frac{\operatorname{tr}(\mathbf{R}) - 1}{2}} \tag{4.20}$$

where

$$\boxed{\text{tr}\,(\mathbf{R}) = R_{11} + R_{22} + R_{33}} \tag{4.21}$$

is the trace of $\mathbf{R}$. This means that there are two possible solutions, Φ_1 and Φ_2, for a given rotation, where Φ_1 and Φ_2 are related as

$$\Phi_2 = \Phi_1 - 2\pi$$

This is because we can rotate about any axis rotation in two directions: clockwise and counterclockwise.

Solving for Φ using the inverse cosine function,

$$\Phi = \arccos\left[\frac{\text{tr}\,(\mathbf{R}) - 1}{2}\right] \tag{4.22}$$

The inverse cosine function will always return a value in the interval $[0, \pi]$. This matches Convention 22. However, let's consider the case where we have constructed some rotation matrix by defining a rotation axis using a principal rotation vector, and then rotating counterclockwise about that axis by $\Phi = 3\pi/2$. If we tried to reconstruct the axis-angle representation from the rotation matrix, Eq. (4.22) would provide us with a positive angle less than π for Φ, even though the principal angle we used to construct the rotation matrix was greater than π. What is happening is the principal angle provided by Eq. (4.25) represents a rotation about a principal rotation vector antiparallel to our original principal rotation vector.

To obtain the principal rotation vector from the rotation matrix, we need to think about what happens if we try to rotate the principal rotation vector itself. If $\mathbf{e}$ is the principal rotation vector defining the axis of rotation, then applying the rotation to $\mathbf{e}$ should not change $\mathbf{e}$.

$$\mathbf{e} = \mathbf{R}\mathbf{e}$$

We can recognize this as the eigenvalue problem

$$\mathbf{R}\mathbf{e} = \lambda\mathbf{e}$$

where the eigenvalue is simply $\lambda = 1$. To solve for the principal rotation vector, $\mathbf{e}$, we simply need to find the eigenvector of $\mathbf{R}$ corresponding to the eigenvalue $\lambda = 1$. However, if we solve the eigenvalue problem numerically, we could get a result that conflicts with Φ. Eq. (4.22) will always provide Φ in the domain $[0, \pi]$, and we need to find the principal rotation vector accordingly; however, the numerical eigenvalue/eigenvector solver would have no knowledge of what quadrant we have returned Φ in. Additionally, a numerical eigenvalue/eigenvector solver can be substantially slower than simple arithmetic operations. Instead, we use an explicit solution for the eigenvector [52, pp. 16–19], [33], [5].

$$\boxed{\mathbf{e} = \frac{1}{2\sin\Phi}\begin{bmatrix} R_{23} - R_{32} \\ R_{31} - R_{13} \\ R_{12} - R_{21} \end{bmatrix}} \tag{4.23}$$

Note that to avoid evaluating $\sin\Phi$, we can replace it with either

$$\boxed{\sin\Phi = \sqrt{1 - \cos^2\Phi}} \tag{4.24}$$

where $\cos\Phi$ is computed using Eq. (4.20) [9, p. 35][2], or with

$$\sin\Phi = \sqrt{[3 - \text{tr}\,(\mathbf{R})][1 + \text{tr}\,(\mathbf{R})]}$$

which can be derived from substituting Eq. (4.20) into the first expression for $\sin\Phi$ above [8]. However, in practice, we will be normalizing the principal rotation vector, so it is useless to compute its coefficient.

Note that when $\Phi = 0$, there is a singularity when computing the Euler axis (see Eq. (4.23)) since $\sin\Phi = 0$. Conceptually, if no rotation occurs, it is impossible to determine the axis of rotation. We *could* just set it equal to any unit vector, but in practice, we have to take more care to ensure that we are consistent across the various rotation parameterizations. More specifically, if we choose $\mathbf{e} = (1, 0, 0)^T$, then the axis-angle to quaternion conversion covered in Section 4.5.6 will automatically yield the identity quaternion (see Section 4.5.2) corresponding to no rotation.

[2] This comes from the fact that $\cos^2\theta + \sin^2\theta = 1$

Convention 24: Singularity handling for the axis-angle representation.

The axis-angle representation has a singularity at $\Phi = 0$, corresponding to no rotation. If we are converting from some other rotation parameterization to the axis-angle representation, we default the principal rotation vector

$$\mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

This allows us to automatically obtain the identity quaternion (see Section 4.5.2) corresponding to no rotation when using Algorithm 27 to convert to a quaternion.

Also, since $\cos 0 = 1$, this corresponds to the right hand side of Eq. (4.20) being equal to 1. However, there is no singularity when $\Phi = \pi$; in that case, there *is* a rotation, so we should be able to determine an axis of rotation. To find the principal rotation vector, we can simply normalize any nonzero column of $\mathbf{R} + \mathbf{I}_{3\times 3}$ [9, pp. 32–37].

Near multiples of π, the derivative of the inverse cosine function approaches infinity, which can lead to numerical issues. To avoid this, we can instead compute the principal angle as [38]

$$\boxed{\Phi = \operatorname{arctan2}(\sin\Phi, \cos\Phi)} \tag{4.25}$$

However, the range of the four-quadrant inverse tangent is $[-\pi, \pi]$, while we want Φ to be in $[0, \pi]$ as before. In the case that we get a negative Φ, we can just add π to it. Since the aforementioned adjustment essentially reverses the direction of rotation, we need to also reverse the direction of the principal rotation vector. Also, note that we can compute $\sin\Phi$ with Eq. (4.24) and $\cos\Phi$ with Eq. (4.20).

Algorithm 12 closely follows the algorithm[3] presented in [9, pp. 32–37, A3–A4], but uses the four-quadrant inverse tangent when computing Φ, instead of the inverse cosine function. Additionally, we do not compute the coefficient of $\mathbf{e}$, since we normalize $\mathbf{e}$ anyways. Note that the 10^{-11} tolerance on the value of $\cos\Phi$ comes from [9, p. A3]. Additionally, due to finite precision arithmetic, $\cos\Phi$ may sometimes be slightly less than -1 or slightly greater than 1, which can cause issues when solving for Φ. Thus, we manually clamp $\cos\Phi$ to have a value between -1 and 1.

Algorithm 12: `mat2axang`
Rotation matrix to axis-angle representation.

Inputs:

- $\mathbf{R} \in \mathbb{R}^{3\times 3}$ - rotation matrix

Procedure:

1. Cosine of the principal angle.

$$c = \frac{R_{11} + R_{22} + R_{33} - 1}{2}$$

2. Ensure that $|c| \leq 1$.

$$c = \max\left[\min\left(c, 1\right), -1\right]$$

3. Edge case #1: $\cos\Phi = 1$.

[3] [9, pp. 40, A4] also presents an additional method for computing the axis-angle representation with superior numerical stability, albeit at the cost of efficiency. However, here, we use their first method, but use the numerically more stable four-quadrant inverse tangent.

if $|c - 1| < 10^{-11}$

4. Principal rotation vector.

$$\mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

5. Principal angle $[\mathrm{rad}]$.

$$\Phi = 0$$

6. Edge case #2: $\cos \Phi = -1$.

else if $|c + 1| < 10^{-11}$

(a) Principal angle $[\mathrm{rad}]$.

$$\Phi = \pi$$

(b) Auxiliary matrix.

$$\mathbf{A} = \mathbf{R} + \mathbf{I}_{3\times 3}$$

(c) Select a nonzero column of $\mathbf{A}$ and store it as $\mathbf{e}$. If no nonzero columns are found, then the principal rotation vector is just the zero vector

$\mathbf{e} = \mathbf{0}$
for $i = 1$ **to** 3
  if $\|\mathbf{A}_{:,i}\| > 10^{-3}$
    $\mathbf{e} = \mathbf{A}_{:,i}$
    break
  end
end

7. Base case.

else

(a) Sine of the principal angle.

$$s = \sqrt{1 - c^2}$$

(b) Principal angle.

$$\Phi = \operatorname{arctan2}(s, c)$$

(c) Principal rotation vector.

$$\mathbf{e} = \begin{bmatrix} R_{23} - R_{32} \\ R_{31} - R_{13} \\ R_{12} - R_{21} \end{bmatrix}$$

(d) Ensure that $\Phi \in [0, \pi]$.

if $\Phi < 0$
  $\Phi = \Phi + \pi$
  $\mathbf{e} = -\mathbf{e}$
end

end

8. Normalize the principal rotation vector.

$$\mathbf{e} = \frac{\mathbf{e}}{\|\mathbf{e}\|}$$

9. Return the results.

return $\mathbf{e}, \Phi$

Outputs:

- $\mathbf{e} \in \mathbb{R}^3$ - principal rotation vector
- $\Phi \in \mathbb{R}$ - principal angle [rad]

Note:

- $\|\mathbf{e}\| = 1$.
- $\Phi \in [0, \pi]$
- If $\Phi = 0$, then $\mathbf{e}$ is returned as $(1, 0, 0)^T$.

Test Cases:

- See Appendix A.1.2.

Rotation Matrix from Axis-Angle Representation

The rotation matrix can be computed from the axis-angle representation as [33], [38]

$$\mathbf{R} = \begin{bmatrix} e_1^2(1-\cos\Phi) + \cos\Phi & e_1e_2(1-\cos\Phi) + e_3\sin\Phi & e_1e_3(1-\cos\Phi) - e_2\sin\Phi \\ e_2e_1(1-\cos\Phi) - e_3\sin\Phi & e_2^2(1-\cos\Phi) + \cos\Phi & e_2e_3(1-\cos\Phi) + e_1\sin\Phi \\ e_3e_1(1-\cos\Phi) + e_2\sin\Phi & e_3e_2(1-\cos\Phi) - e_1\sin\Phi & e_3^2(1-\cos\Phi) + \cos\Phi \end{bmatrix} \tag{4.26}$$

Algorithm 13: `axang2mat`
Axis-angle representation to rotation matrix.

Inputs:
- $\mathbf{e} \in \mathbb{R}^3$ - principal rotation vector
- $\Phi \in \mathbb{R}$ - principal angle [rad]

Procedure:
1. Normalize the principal rotation vector.

$$\mathbf{e} = \frac{\mathbf{e}}{\|\mathbf{e}\|}$$

2. Precompute the trigonometric functions.

$$c = \cos\Phi$$
$$s = \sin\Phi$$

3. Auxiliary parameter.

$$a = 1 - c$$

4. Rotation matrix.

$$\mathbf{R} = \begin{bmatrix} e_1^2 a + c & e_1 e_2 a + e_3 s & e_1 e_3 a - e_2 s \\ e_2 e_1 a - e_3 s & e_2^2 a + c & e_2 e_3 a + e_1 s \\ e_3 e_1 a + e_2 s & e_3 e_2 a - e_1 s & e_3^2 a + c \end{bmatrix}$$

5. Return the result.

return $\mathbf{R}$

Outputs:
- $\mathbf{R} \in \mathbb{R}^{3\times 3}$ - rotation matrix

Note:
- This algorithm normalizes $\mathbf{e}$.

Test Cases:
- See Appendix A.1.2.

4.4.2 Relationship With Euler Angles

Recall Eqs. (4.17), (4.18), and (4.19) defining the 3-2-1 Euler angles in terms of the elements of the rotation matrix. Also, recall Eq. (4.26) defining the elements of the rotation matrix in terms of the axis-angle representation. Using these equations, we can obtain the 3-2-1 Euler angles corresponding to an axis-angle representation [6].

$$\boxed{\psi = \begin{cases} \operatorname{arctan2}\left[e_1 e_2 (1-\cos\Phi) + e_3 \sin\Phi, e_1^2(1-\cos\Phi) + \cos\Phi\right], & |R_{13}| < 1 \\ 0, & |R_{13}| \geq 1 \end{cases}} \tag{4.27}$$

$$\boxed{\theta = \begin{cases} -\arcsin R_{13}, & |R_{13}| < 1 \\ -\left(\frac{\pi}{2}\right) \operatorname{sgn} R_{13}, & |R_{13}| \geq 1 \end{cases}} \tag{4.28}$$

$$\boxed{\phi = \begin{cases} \operatorname{arctan2}\left[e_2 e_3(1-\cos\Phi) + e_1 \sin\Phi, e_3^2(1-\cos\Phi) + \cos\Phi\right], & |R_{13}| < 1 \\ \operatorname{arctan2}\left[e_2 e_1(1-\cos\Phi) - e_3 \sin\Phi, e_3 e_1(1-\cos\Phi) + e_2 \sin\Phi\right], & R_{13} \leq -1 \\ -\operatorname{arctan2}\left[e_3 e_2(1-\cos\Phi) - e_1 \sin\Phi, e_2^2(1-\cos\Phi) + \cos\Phi\right], & R_{13} \geq 1 \end{cases}} \tag{4.29}$$

where

$$\boxed{R_{13} = e_1 e_3(1-\cos\Phi) - e_2 \sin\Phi} \tag{4.30}$$

Algorithm 14 below is identical in logic to that of Algorithm 11. In principle, we are converting the axis-angle representation to a rotation matrix, and then extracting the 3-2-1 Euler angles from the rotation matrix.

Algorithm 14: `axang2eul_321`

Axis-angle representation to 3-2-1 Euler angles (yaw, pitch, and roll).

Inputs:

- $\mathbf{e} \in \mathbb{R}^3$ - principal rotation vector
- $\Phi \in \mathbb{R}$ - principal angle [rad]

Procedure:

1. Normalize the principal rotation vector.

$$\mathbf{e} = \frac{\mathbf{e}}{\|\mathbf{e}\|}$$

2. Precompute the trigonometric functions.

$$c = \cos\Phi$$
$$s = \sin\Phi$$

3. Store the elements of the principal rotation vector, $\mathbf{e} = (e_1, e_2, e_3)^T$, individually.
4. Calculate the R_{13} element of the rotation matrix.

$$R_{13} = e_1 e_3(1-c) - e_2 s$$

5. Booleans that store whether or not we are in a singular case, and if so, which one.

$$\texttt{pitch\_up} = (R_{13} \leq -1)$$
$$\texttt{pitch\_down} = (R_{13} \geq 1)$$
$$\texttt{singular} = (\texttt{pitch\_up}\ \textbf{or}\ \texttt{pitch\_down})$$

6. Yaw angle [rad].

if `singular`
$\quad \psi = 0$
else
$\quad \psi = \operatorname{arctan2}\left[e_1 e_2(1-c) + e_3 s, e_1^2(1-c) + c\right]$
end

7. Pitch angle $[\text{rad}]$.

   if `singular`
   $\quad \theta = -\left(\frac{\pi}{2}\right) \operatorname{sgn} R_{13}$
   else
   $\quad \theta = -\arcsin R_{13}$
   end

8. Roll angle $[\text{rad}]$.

   if `pitch_up`
   $\quad \phi = \operatorname{arctan2}\left[e_2 e_1(1-c) - e_3 s, e_3 e_1(1-c) + e_2 s\right]$
   else if `pitch_down`
   $\quad \phi = -\operatorname{arctan2}\left[e_3 e_2(1-c) - e_1 s, e_2^2(1-c) + c\right]$
   else
   $\quad \phi = \operatorname{arctan2}\left[e_2 e_3(1-c) + e_1 s, e_3^2(1-c) + c\right]$
   end

9. Return the results.

   return ψ, θ, ϕ

Outputs:

- $\psi \in \mathbb{R}$ - yaw angle (1st rotation, about 3rd axis) $[\text{rad}]$
- $\theta \in \mathbb{R}$ - pitch angle (2nd rotation, about 2nd axis) $[\text{rad}]$
- $\phi \in \mathbb{R}$ - roll angle (3rd rotation, about 1st axis) $[\text{rad}]$

Note:

- This algorithm normalizes $\mathbf{e}$.
- $\psi \in (-\pi, \pi]$
- $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$
- $\phi \in (-\pi, \pi]$
- When a singularity corresponding to the pitch angles of $\theta = \pm\pi/2$ is encountered, the yaw angle (ψ) is set to 0 and the roll angle (ϕ) is determined accordingly.

Test Cases:

- See Appendix A.1.2.

As we go through all these rotation parameterizations and the conversions between them, note that we write all conversions in an explicit form, i.e. not depending on an intermediate conversion. The one exception is the conversion from 3-2-1 Euler angles to the axis-angle representation. For this conversion, we actually skip ahead to the material covered in Section 4.5.5 and first convert the 3-2-1 Euler angles to a quaternion using Algorithm 26. We then obtain the axis-angle representation from the quaternion using Algorithm 28.

Algorithm 15: `eul2axang_321`

3-2-1 Euler angles (yaw, pitch, and roll) to axis-angle representation.

Inputs:

- $\psi \in \mathbb{R}$ - yaw angle (1st rotation, about 3rd axis) [rad]
- $\theta \in \mathbb{R}$ - pitch angle (2nd rotation, about 2nd axis) [rad]
- $\phi \in \mathbb{R}$ - roll angle (3rd rotation, about 1st axis) [rad]

Procedure:

1. Unit quaternion (Algorithm 26).

$$\mathbf{q} = \texttt{eul2quat\_321}(\psi, \theta, \phi)$$

2. Principal rotation vector and principal angle [rad] (Algorithm 28).

$$[\mathbf{e}, \Phi] = \texttt{quat2axang}(\mathbf{q})$$

3. Return the results.

return ψ, θ, ϕ

Outputs:

- $\mathbf{e} \in \mathbb{R}^3$ - principal rotation vector
- $\Phi \in \mathbb{R}$ - principal angle [rad]

Note:

- $\|\mathbf{e}\| = 1$.
- $\Phi \in [0, \pi]$
- If $\Phi = 0$, then $\mathbf{e}$ is returned as $(1, 0, 0)^T$.

Test Cases:

- See Appendix A.1.2.

4.4.3 Disadvantages

The axis-angle representation is not unique, since a rotation of $-\Phi$ about $-\mathbf{e}$ is the same as a rotation of Φ about $\mathbf{e}$ [5]. Additionally, it is not possible to determine a principal rotation vector for a principal angle of 0. Finally, there is an ambiguity in the direction of the principal rotation vector at $\Phi = \pi$.

4.5 Quaternions

A **quaternion** is a four-dimensional vector that can be used to parameterize a rotation. It consists of a scalar component, q_0, and a vector component, $\mathbf{q}_v = (q_1, q_2, q_3)^T$.

$$\boxed{\mathbf{q} = \begin{bmatrix} q_0 \\ \mathbf{q}_v \end{bmatrix} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}} \tag{4.31}$$

Convention 25: Scalar-first convention.

Throughout this text, we use the **scalar-first convention**, where the scalar component of the quaternion is the first element of the quaternion, as shown in Eq. (4.31). This is the most widely-used convention in aerospace.

In some cases, the scalar-*last* convention is used, where the quaternion is defined as $\mathbf{q} = (q_1, q_2, q_3, q_4)^T$, where q_4 is the scalar component and where $\mathbf{q}_v = (q_1, q_2, q_3)^T$ is still the vector component.

In general, quaternions do not have to have unit magnitude. However, when used for rotation parameterization, we deal specifically with **unit quaternions**. Unit quaternions are covered in more detail in Section 4.5.2.

We will be referring to quaternions and unit quaternions interchangeably, but typically, when we refer to a quaternion in this text, we are referring to a *unit* quaternion. However, the algorithms that take a quaternion and convert it to an alternate representation do not make the assumption that we are dealing with unit quaternions; they will normalize the quaternions by default.

Like the axis-angle representation, unit quaternions have a choice of sign[4]; that is, for the purposes of representing rotations, $\mathbf{q}$ is equivalent to $-\mathbf{q}$.

$$\boxed{\mathbf{q} \equiv -\mathbf{q}} \tag{4.32}$$

Due to this freedom in picking a sign, we have to select a convention. We choose for the scalar component to be positive.

Convention 26: Quaternion sign convention.

The scalar component of a quaternion, q_0, should be positive.

See Section 4.5.6 for more information on the relationship between unit quaternions and the axis-angle representation.

In practice, it is essential to subscript the unit quaternions with what rotation they represent. For example, if we are transforming the coordinates of a vector from frame A to frame B, and are representing that rotation using a unit quaternion, then we would label that unit quaternion as $\mathbf{q}_{A\to B}$.

4.5.1 Treatment as a Column Vector

The mathematical details of quaternions require a greater amount of mathematical maturity than rotation matrices and Euler angles. These details are generally covered in greater detail in most other texts, but they do not often make it clear how to *use* quaternions. Here, we focus simply on the implementation side.

The main thing we *do* want to note is regarding quaternions is that the set of quaternions forms its own vector space, $\mathbb{H}$; quaternions are ***not*** members of the vector space $\mathbb{R}^4$. In a more traditional mathematical sense, quaternions

[4] It is incorrect to refer to this as a sign ambiguity, since regardless of the sign you choose, there is not ambiguity as to which rotation the unit quaternion represents

are typically defined as

$$\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$$

where

$$q_0, q_1, q_2, q_3 \in \mathbb{R}$$

and where $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are symbols that can be interpreted as unit vectors point along three spatial axes. Note that this is very similar to how we can write a complex number, $z \in \mathbb{C}$, as

$$z = a + bi$$

where $a, b \in \mathbb{R}$ and where i is the imaginary unit. Just like how complex numbers are often analogous to vectors in $\mathbb{R}^2$ (i.e. they are plotted as vectors in the imaginary plane), we can often treat a quaternion like it is a vector in $\mathbb{R}^4$ [35]. This view of quaternions is also suggested by how we initially defined the quaternion as

$$\mathbf{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

in Eq. (4.31). **Thus, in most practical applications, we simply treat quaternions as column vectors (with some caveats).**

For example, we take the transpose of a quaternion as

$$\mathbf{q}^T = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}$$

so the inner product of two quaternions $\mathbf{p}, \mathbf{q} \in \mathbb{H}$ is

$$\mathbf{p}^T\mathbf{q} = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = p_0q_0 + p_1q_1 + p_2q_2 + p_3q_3$$

4.5.2 Quaternion Properties and Definitions

Conjugate

The **conjugate** of a quaternion is defined as [13, 35]

$$\boxed{\mathbf{q}^* = \begin{bmatrix} q_0 \\ -\mathbf{q}_v \end{bmatrix} = \begin{bmatrix} q_0 \\ -q_1 \\ -q_2 \\ -q_3 \end{bmatrix}} \tag{4.33}$$

Algorithm 16: `quatconj`
Conjugate of a quaternion.

Inputs:

- $\mathbf{q} \in \mathbb{H}$ - quaternion

Procedure:

$$\mathbf{q}^* = \begin{bmatrix} q_0 \\ -q_1 \\ -q_2 \\ -q_3 \end{bmatrix}$$

return $\mathbf{q}^*$

Outputs:

- $\mathbf{q}^{-1} \in \mathbb{H}$ - quaternion conjugate

Note:

- This algorithm assumes the scalar-first convention for quaternions.
- $\mathbf{q}$ does not have to be input as a unit quaternion (this algorithm is for all quaternions, not just unit quaternions).
- $\mathbf{q}^*$ is not normalized.

Test Cases:

- See Appendix A.1.3.

Multiplication

The **Hamilton product** (multiplication) of two quaternions, $\mathbf{p} = (p_0, p_1, p_2, p_3)^T$ and $\mathbf{q} = (q_0, q_1, q_2, q_3)^T$, is defined as

$$\boxed{\mathbf{p} \otimes \mathbf{q} = \begin{bmatrix} p_0 q_0 - p_1 q_1 - p_2 q_2 - p_3 q_3 \\ p_0 q_1 + p_1 q_0 + p_2 q_3 - p_3 q_2 \\ p_0 q_2 - p_1 q_3 + p_2 q_0 + p_3 q_1 \\ p_0 q_3 + p_1 q_2 - p_2 q_1 + p_3 q_0 \end{bmatrix}} \tag{4.34}$$

Note that like matrix multiplication, quaternion multiplication is *not* commutative [35, 39].

$$\mathbf{p} \otimes \mathbf{q} \neq \mathbf{q} \otimes \mathbf{p}$$

However, quaternions are associative over multiplication [48, p. 47]:

$$\boxed{(\mathbf{p} \otimes \mathbf{q}) \otimes \mathbf{r} = \mathbf{p} \otimes (\mathbf{q} \otimes \mathbf{r})} \tag{4.35}$$

Algorithm 17: `quatmul`
Quaternion multiplication (Hamilton product).

Inputs:

- $\mathbf{p} \in \mathbb{H}$ - quaternion
- $\mathbf{q} \in \mathbb{H}$ - quaternion

Procedure:

1. Evaluate the Hamilton product.

$$\mathbf{r} = \begin{bmatrix} p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3 \\ p_0q_1 + p_1q_0 + p_2q_3 - p_3q_2 \\ p_0q_2 - p_1q_3 + p_2q_0 + p_3q_1 \\ p_0q_3 + p_1q_2 - p_2q_1 + p_3q_0 \end{bmatrix}$$

2. Ensure that the scalar component of the quaternion is positive.

if $r_0 < 0$
  $\mathbf{r} = -\mathbf{r}$
end

3. Return the result.

return $\mathbf{r}$

Outputs:

- $\mathbf{r} \in \mathbb{H}$ - Hamilton product of $\mathbf{p}$ and $\mathbf{q}$

Note:

- This algorithm assumes the scalar-first convention for quaternions.
- $\mathbf{p}$ and $\mathbf{q}$ do not have to be input as unit quaternions (this algorithm is for all quaternions, not just unit quaternions).
- $\mathbf{r}$ is not normalized.
- The scalar component of $\mathbf{r}$ is chosen to be positive.

Test Cases:

- See Appendix A.1.3.

Norm

The **norm** of a quaternion (representing its magnitude) is defined as

$$\|\mathbf{q}\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

Treating the quaternion as a column vector, we can write its norm as

$$\boxed{\|\mathbf{q}\| = \sqrt{\mathbf{q}^T\mathbf{q}} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}} \tag{4.36}$$

This is analogous to the 2-norm of a vector in $\mathbb{R}^4$ [35], [23, p. 22].

A **unit quaternion** is a quaternion with a norm of 1.

Algorithm 18: `quatnorm`
Norm of a quaternion.

Inputs:

- $\mathbf{q} \in \mathbb{H}$ - quaternion

Procedure:

$$\|\mathbf{q}\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

return $\|\mathbf{q}\|$

Outputs:

- $\|\mathbf{q}\| \in \mathbb{H}$ - norm of the quaternion

Test Cases:

- See Appendix A.1.3.

In many programming languages, it is a bit redundant to implement Algorithm 18; for example, we could just use MATLAB's built-in `norm` function, which calculates the 2-norm of a vector by default. However, it is still useful to implement this algorithm so that "under-the-hood" we can replace how we compute $\|\mathbf{q}\|$ is needed.

Inverse

The **inverse** of a quaternion is just its conjugate divided by its norm squared [13], [35], [23, p. 22].

$$\boxed{\mathbf{q}^{-1} = \frac{\mathbf{q}^*}{\|\mathbf{q}\|^2} = \frac{\mathbf{q}^*}{\mathbf{q}^T\mathbf{q}} = \frac{\mathbf{q}^*}{q_0^2 + q_1^2 + q_2^2 + q_3^2}} \tag{4.37}$$

Note that the inverse of a unit quaternion is just its conjugate (since its norm is 1).

Algorithm 19: `quatinv`
Inverse of a quaternion.

Inputs:

- $\mathbf{q} \in \mathbb{H}$ - quaternion

Procedure:

$$\mathbf{q}^{-1} = \frac{\texttt{quatconj}(\mathbf{q})}{\mathbf{q}^T\mathbf{q}} \quad \text{(Algorithm 16)}$$

return $\mathbf{q}^{-1}$

Outputs:

- $\mathbf{q}^* \in \mathbb{H}$ - quaternion inverse

Note:

- This algorithm assumes the scalar-first convention for quaternions.
- $\mathbf{q}$ does not have to be input as a unit quaternion (this algorithm is for all quaternions, not just unit quaternions).
- $\mathbf{q}^{-1}$ is not normalized.

Test Cases:

- See Appendix A.1.3.

Inverse of a Product

The inverse of the Hamilton product of two quaternions is given by the Hamilton product of their inverses, in reverse order (similar to the inverse of the product of two square matrices) [48, p. 48].

$$\boxed{(\mathbf{p} \otimes \mathbf{q})^{-1} = \mathbf{q}^{-1} \otimes \mathbf{p}^{-1}} \tag{4.38}$$

Identity Quaternion

A quaternion multiplied by its inverse is equal to the **identity quaternion**. The identity quaternion is the unit quaternion with scalar part equal to 1 and vector part equal to $\mathbf{0}$. Note that the Hamilton product of a quaternion with its inverse[5] yields the identity quaternion [23, p. 22].

$$\boxed{\mathbf{q}^{-1} \otimes \mathbf{q} = \mathbf{q} \otimes \mathbf{q}^{-1} = \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}} \tag{4.39}$$

which results in no rotation.

Normalization

In aerospace simulations, quaternions will often form part of the state vector that is being numerically integrated. Since a standard numerical integrator won't automatically preserve the unit norm constraint on unit quaternion, we will want to regularly renormalize the quaternion. This can be done by simply dividing the quaternion by its norm.

$$\boxed{\mathbf{q} = \frac{\mathbf{q}}{\|\mathbf{q}\|}} \tag{4.40}$$

Since we will need to normalize the quaternion often, we introduce Algorithm 20 in order to reduce boilerplate code.

> In general, we will assume the quaternions input to any algorithms are already normalized; it is *not* the responsibility of an algorithm to normalize its inputs. However, it *is* the responsibility of an algorithm to normalize its *outputs*.

Algorithm 20: `quatnormalize`
Normalize a quaternion.

Inputs:

- $\mathbf{q} \in \mathbb{H}$ - quaternion

Procedure:

$\mathbf{q} = \dfrac{\mathbf{q}}{\texttt{quatnorm}(\mathbf{q})}$ using Algorithm 18

return $\mathbf{q}$

[5] Hence, the quaternion inverse we define is a *multiplicative* inverse

Outputs:

- $\mathbf{q} \in \mathbb{H}$ - unit quaternion

Test Cases:

- See Appendix A.1.3.

4.5.3 Rotations via Unit Quaternions

The rotation from coordinate frame A to coordinate frame B can be parameterized as the unit quaternion

$$\mathbf{q}_{A\to B}$$

Since a unit quaternion multiplied by its inverse results in no rotation, and since the inverse of a unit quaternion is just its conjugate, we know [48, p. 50]

$$\boxed{\mathbf{q}_{B\to A} = \mathbf{q}_{A\to B}^{-1} = \mathbf{q}_{A\to B}^{*}} \tag{4.41}$$

Applying a Single Rotation

Let $[\boldsymbol{r}]_A$ be a vector expressed in frame A. Let

$$\boxed{\mathbf{p}_A = \begin{pmatrix} 0 \\ [\boldsymbol{r}]_A \end{pmatrix}} \tag{4.42}$$

To express this vector in frame B[6] [48, p. 50],

$$\boxed{\mathbf{p}_B = \mathbf{q}_{A\to B}^{-1} \otimes \mathbf{p}_A \otimes \mathbf{q}_{A\to B}} \tag{4.43}$$

where

$$\boxed{\mathbf{p}_B = \begin{pmatrix} 0 \\ [\boldsymbol{r}]_B \end{pmatrix}} \tag{4.44}$$

If we let

$$\mathbf{q}_{A\to B} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

then using the definition of the Hamilton product from Eq. (4.34), we can simply Eq. (4.43) to [36]

$$[\boldsymbol{r}]_B = \underbrace{\begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix}}_{\mathbf{R}_{A\to B}} [\boldsymbol{r}]_A$$

[6] [36] provides this equation as

$$[\boldsymbol{r}]_B = \mathbf{q}_{A\to B} \otimes \mathbf{p}_A \otimes \mathbf{q}_{A\to B}^{-1}$$

, but from the discussion in that article, this equation appears to encode an active rotation, rather than the passive rotations we use in aerospace. Even more confusingly, [23, p. 34], which uses passive rotations throughout almost all rotation discussions, also uses this active version of the quaternion rotation. However, [39] does clearly indicate the active vs. passive formulations, and [48, p. 50] also defines *and* describes the passive formulation.

In practice, we first convert the quaternion to a rotation matrix (see Algorithm 23 in Section 4.5.4), and then perform the rotation using that rotation matrix. This results in fewer total arithemtic operations than using Algorithm 17 twice to evaluate Eq. (4.43) directly [13], [36].

Algorithm 21: `quatrotate`
Passive rotation of a vector by a unit quaternion.

Inputs:
- $[\mathbf{q}_{A\to B} \in \mathbb{H}$ - unit quaternion representing rotation from frame A to frame B
- $[\boldsymbol{r}]_A \in \mathbb{R}^3$ - vector expressed in coordinate frame A

Procedure:

$[\boldsymbol{r}]_B = \texttt{matrotate}\,(\texttt{quat2mat}(\mathbf{q}_{A\to B}), [\boldsymbol{r}]_A)$ (Algorithms 8 and 23)
return $[\boldsymbol{r}]_B$

Outputs:
- $[\boldsymbol{r}]_B \in \mathbb{R}^3$ - vector expressed in coordinate frame B

Note:
- This algorithm assumes the scalar-first convention for quaternions.
- This algorithm normalizes the input quaternion.

Test Cases:
- See Appendix A.1.3.

Chaining Rotations

Let's define the following three unit quaternions:

1. $q_{A\to B}$: unit quaternion representing the rotation from frame A to frame B
2. $q_{B\to C}$: unit quaternion representing the rotation from frame B to frame C
3. $q_{A\to C}$: unit quaternion representing the rotation from frame A to frame C

Recall that for rotation matrices, the rotations chain together right-to-left in the order of application:

$$\mathbf{R}_{A\to C} = \mathbf{R}_{B\to C}\mathbf{R}_{A\to B}$$

For unit quaternions, the rotations chain together in the opposite order, from left-to-right [13, 36]:

$$\boxed{\mathbf{q}_{A\to C} = (\mathbf{q}_{A\to B}) \otimes (\mathbf{q}_{B\to C})} \tag{4.45}$$

This can be shown by first applying the rotation from A to B using Eq. (4.43).

$$\mathbf{p}_B = \mathbf{q}_{A\to B}^{-1} \otimes \mathbf{p}_A \otimes \mathbf{q}_{A\to B}$$

Next, using Eq. (4.43) once more to apply the rotation from B to C,

$$\begin{aligned}
\mathbf{p}_C &= \mathbf{q}_{B\to C}^{-1} \otimes \mathbf{p}_B \otimes \mathbf{q}_{B\to C} \\
&= \mathbf{q}_{B\to C}^{-1} \otimes (\mathbf{q}_{A\to B}^{-1} \otimes \mathbf{p}_A \otimes \mathbf{q}_{A\to B}) \otimes \mathbf{q}_{B\to C} \\
&= (\mathbf{q}_{B\to C}^{-1} \otimes \mathbf{q}_{A\to B}^{-1}) \otimes \mathbf{p}_A \otimes (\mathbf{q}_{A\to B} \otimes \mathbf{q}_{B\to C}) \\
&= (\mathbf{q}_{A\to B} \otimes \mathbf{q}_{B\to C})^{-1} \otimes \mathbf{p}_A \otimes (\mathbf{q}_{A\to B} \otimes \mathbf{q}_{B\to C})
\end{aligned} \tag{4.46}$$

Note that the simplification above relies on Eqs. (4.38) and (4.35). From Eq. (4.43), we also know

$$\mathbf{p}_C = \mathbf{q}_{A\to C}^{-1} \otimes \mathbf{p}_A \otimes \mathbf{q}_{A\to C} \tag{4.47}$$

Comparing Eqs. (4.46) and (4.47), we can see that

$$\mathbf{q}_{A\to C} = (\mathbf{q}_{A\to B}) \otimes (\mathbf{q}_{B\to C})$$

as given previously by Eq. (4.45) [48, p. 50], [39].

Due to floating point errors, the Hamilton product of two unit quaternions could have a magnitude that is slightly different than 1. Since in this context we are using quaternions exclusively for parameterizing rotations, we normalize the Hamilton product of two unit quaternions to ensure that the result is a unit quaternion.

Algorithm 22: `quatchain`
Chaining passive rotations represented by unit quaternions.

Inputs:
- $\mathbf{q}_{A\to B} \in \mathbb{H}$ - unit quaternion representing rotation from frame A to frame B
- $\mathbf{q}_{B\to C} \in \mathbb{H}$ - unit quaternion representing rotation from frame B to frame C

Procedure:
1. Evaluate the Hamilton product to find the chained rotation (Algorithm 17).

   $$\mathbf{q}_{A\to C} = \texttt{quatmul}(\mathbf{q}_{A\to B}, \mathbf{q}_{B\to C})$$

2. Normalize the result (Algorithm 20).

   $$\mathbf{q}_{A\to C} = \texttt{quatnormalize}(\mathbf{q}_{A\to C})$$

3. Return the result.

   return $\mathbf{q}_{A\to C}$

Outputs:
- $\mathbf{q}_{A\to C} \in \mathbb{H}$ - unit quaternion representing rotation from frame A to frame C

Note:
- This algorithm assumes the scalar-first convention for quaternions.
- This algorithm assumes that $\mathbf{q}_{A\to B}$ and $\mathbf{q}_{B\to C}$ are input as unit quaternions, so it does *not* normalize them.
- $\mathbf{q}_{A\to C}$ is normalized.
- The scalar component of $\mathbf{q}_{A\to C}$ is chosen to be positive.

Test Cases:
- See Appendix A.1.3.

4.5.4 Relationship With Rotation Matrices

Recall that we can take a vector expressed in frame A and resolve it in frame B using a rotation matrix.

$$[\mathbf{v}]_B = \mathbf{R}_{A\to B}[\mathbf{v}]_A$$

Let $\mathbf{q}$ be a quaternion representing a rotation. The corresponding rotation matrix, $\mathbf{R}$, can be determined as [36], [23, p. 23]

$$\boxed{\mathbf{R} = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix}} \tag{4.48}$$

If we are storing a rotation as a quaternion, we first convert the quaternion to a rotation matrix using Algorithm 23 below. We can then apply the rotation using the rotation matrix.

Algorithm 23: `quat2mat`
Unit quaternion to rotation matrix.

Inputs:

- $\mathbf{q} \in \mathbb{H}$ - unit quaternion

Procedure:

1. Normalize the quaternion (Algorithm 20).

   $$\mathbf{q} = \texttt{quatnormalize}(\mathbf{q})$$

2. Store the elements of the unit quaternion, $\mathbf{q} = (q_0, q_1, q_2, q_3)^T$, individually.
3. Construct the rotation matrix.

   $$\mathbf{R} = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix}$$

4. Return the result.

   return $\mathbf{R}$

Outputs:

- $\mathbf{R} \in \mathbb{R}^{3\times 3}$ - rotation matrix

Note:

- This algorithm assumes the scalar-first convention for quaternions.
- This algorithm normalizes the input quaternion.

Test Cases:

- See Appendix A.1.2.

Extracting a quaternion from a rotation matrix is a more involved process since there are many alternative ways that each quaternion element can be computed. While all alternatives are algebraically equivalent, they are not numerically equivalent. As an example, one way we could compute that elements would be to first compute q_1 as

$$q_1 = \pm\frac{1}{2}\sqrt{1+R_{11}-R_{22}-R_{33}}$$

and then compute the remaining elements as

$$q_0 = \frac{R_{23} - R_{32}}{4q_1}$$
$$q_2 = \frac{R_{12} + R_{21}}{4q_1}$$
$$q_3 = \frac{R_{31} + R_{13}}{4q_1}$$

However, if q_1 is near 0, then the problem is ill-conditioned mathematically. The most popular and common method is **Shepperd's method**, which first computes each quaternion element with an equation similar to the one for q_1, picks the element that has the largest value, and then computes the remaining elements using equations similar to the ones for q_0, q_2, and q_3 above. Shepperd's method is covered in more detail in [52, pp. 24–26], [40], [41], and [4].

While Shepperd's method considers 4 alternatives, [41] introduced the Sarabandi-Thomas method in 2019 that considers 16 alternatives, and is (a) more efficient computationally and (b) is more accurate. In that same year, Sarabandi and Thomas also performed a survey on the computation of quaternions from rotation matrices in [40] and determined that the best method is **Cayley's method**. This method is the fastest, has the lowest average error, and is the simplest to implement (the other methods typically require either computing multiple solutions and using a voting scheme, or using if/else statements when computing each quaternion element. Note that the worst case error with Cayley method was slightly higher than that of the Sarabandi-Thomas and Shepperd methods.

Cayley's method computes the unit quaternion as

$$\mathbf{q} = \frac{1}{4}\begin{bmatrix} \sqrt{(R_{11}+R_{22}+R_{33}+1)^2 + (R_{32}-R_{23})^2 + (R_{13}-R_{31})^2 + (R_{21}-R_{12})^2} \\ \operatorname{sgn}(R_{23}-R_{32})\sqrt{(R_{32}-R_{23})^2 + (R_{11}-R_{22}-R_{33}+1)^2 + (R_{21}+R_{12})^2 + (R_{31}+R_{13})^2} \\ \operatorname{sgn}(R_{31}-R_{13})\sqrt{(R_{13}-R_{31})^2 + (R_{21}+R_{12})^2 + (R_{22}-R_{11}-R_{33}+1)^2 + (R_{32}+R_{23})^2} \\ \operatorname{sgn}(R_{12}-R_{21})\sqrt{(R_{21}-R_{12})^2 + (R_{31}+R_{13})^2 + (R_{32}+R_{23})^2 + (R_{33}-R_{11}-R_{22}+1)^2} \end{bmatrix} \tag{4.49}$$

Note that the scalar component already follows the quaternion sign convention defined by Convention 26 (i.e. $q_0 \geq 0$). Additionally, note that we the signs on the vector component are reversed from what [40] defines[7]. We implement Cayley's method for computing the quaternion from a rotation matrix in Algorithm 24 below.

Algorithm 24: `mat2quat`
Rotation matrix to unit quaternion.

Inputs:

- $\mathbf{R} \in \mathbb{R}^{3\times3}$ - rotation matrix

Procedure:

1. Compute the quaternion using Eq. (4.49). Note that we don't need to include the coefficient of $1/4$ since we will be normalizing in the next step anyways.
2. Normalize the quaternion (Algorithm 20).

   $\mathbf{q} = \texttt{quatnormalize}(\mathbf{q})$

3. Return the result.

   return $\mathbf{q}$

Outputs:

- $\mathbf{q} \in \mathbb{H}$ - unit quaternion

[7] This flip in sign was determined through testing Algorithm 24.

Note:
- This algorithm assumes the scalar-first convention for quaternions.
- The scalar component, q_0, is chosen to be positive.

Test Cases:
- See Appendix A.1.2.

4.5.5 Relationship With Euler Angles

Recall Eqs. (4.17), (4.18), and (4.19) defining the 3-2-1 Euler angles in terms of the elements of the rotation matrix. Also, recall Eq. (4.48) defining the elements of the rotation matrix in terms of a unit quaternion. Using these equations, we can obtain the 3-2-1 Euler angles corresponding to a unit quaternion [23, p. 26]

$$\boxed{\psi = \begin{cases} \arctan2\left[2(q_1q_2 + q_0q_3), 1 - 2(q_2^2 + q_3^2)\right], & |R_{13}| < 1 \\ 0, & |R_{13}| \geq 1 \end{cases}} \tag{4.50}$$

$$\boxed{\theta = \begin{cases} -\arcsin R_{13}, & |R_{13}| < 1 \\ -\left(\frac{\pi}{2}\right) \operatorname{sgn} R_{13}, & |R_{13}| \geq 1 \end{cases}} \tag{4.51}$$

$$\boxed{\phi = \begin{cases} \arctan2\left[2(q_2q_3 + q_0q_1), 1 - 2(q_1^2 + q_2^2)\right], & |R_{13}| < 1 \\ \arctan2\left[2(q_1q_2 - q_0q_3), 2(q_1q_3 + q_0q_2)\right], & R_{13} \leq -1 \\ -\arctan2\left[2(q_2q_3 - q_0q_1), 1 - 2(q_1^2 + q_3^2)\right], & R_{13} \geq 1 \end{cases}} \tag{4.52}$$

where

$$\boxed{R_{13} = 2(q_1q_3 - q_0q_2)} \tag{4.53}$$

Algorithm 25 below is identical in logic to that of Algorithm 11. In principle, we are converting the quaternion to a rotation matrix, and then extracting the 3-2-1 Euler angles from the rotation matrix.

Algorithm 25: `quat2eul_321`
Unit quaternion to 3-2-1 Euler angles (yaw, pitch, and roll).

Inputs:
- $\mathbf{q} \in \mathbb{H}$ - unit quaternion

Procedure:
1. Normalize the quaternion (Algorithm 20).

   $$\mathbf{q} = \texttt{quatnormalize}(\mathbf{q})$$

2. Store the elements of the unit quaternion, $\mathbf{q} = (q_0, q_1, q_2, q_3)^T$, individually.
3. Calculate the R_{13} element of the rotation matrix.

   $$R_{13} = 2(q_1q_3 - q_0q_2)$$

4. Booleans that store whether or not we are in a singular case, and if so, which one.

   $$\texttt{pitch\_up} = (R_{13} \leq -1)$$
   $$\texttt{pitch\_down} = (R_{13} \geq 1)$$
   $$\texttt{singular} = (\texttt{pitch\_up}\ \textbf{or}\ \texttt{pitch\_down})$$

5. Yaw angle [rad].

if `singular`
$\quad \psi = 0$
else
$\quad \psi = \operatorname{arctan2}\left[2(q_1 q_2 + q_0 q_3), 1 - 2(q_2^2 + q_3^2)\right]$
end

6. Pitch angle [rad].

if `singular`
$\quad \theta = -\left(\frac{\pi}{2}\right) \operatorname{sgn} R_{13}$
else
$\quad \theta = -\arcsin R_{13}$
end

7. Roll angle [rad].

if `pitch_up`
$\quad \phi = \operatorname{arctan2}\left[2(q_1 q_2 - q_0 q_3), 2(q_1 q_3 + q_0 q_2)\right]$
else if `pitch_down`
$\quad \phi = -\operatorname{arctan2}\left[2(q_2 q_3 - q_0 q_1), 1 - 2(q_1^2 + q_3^2)\right]$
else
$\quad \phi = \operatorname{arctan2}\left[2(q_2 q_3 + q_0 q_1), 1 - 2(q_1^2 + q_2^2)\right]$
end

8. Return the results.

return ψ, θ, ϕ

Outputs:

- $\psi \in \mathbb{R}$ - yaw angle (1st rotation, about 3rd axis) [rad]
- $\theta \in \mathbb{R}$ - pitch angle (2nd rotation, about 2nd axis) [rad]
- $\phi \in \mathbb{R}$ - roll angle (3rd rotation, about 1st axis) [rad]

Note:

- This algorithm assumes the scalar-first convention for quaternions.
- This algorithm normalizes the input quaternion.
- $\psi \in (-\pi, \pi]$
- $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$
- $\phi \in (-\pi, \pi]$
- When a singularity corresponding to the pitch angles of $\theta = \pm\pi/2$ is encountered, the yaw angle (ψ) is set to 0 and the roll angle (ϕ) is determined accordingly.

Test Cases:

- See Appendix A.1.2.

A unit quaternion can be constructed from the 3-2-1 Euler angles as [48, p. 52], [15, p. 12]

$$\mathbf{q} = \begin{bmatrix} \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) - \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) - \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) \end{bmatrix} \tag{4.54}$$

Algorithm 26: `eul2quat_321`

3-2-1 Euler angles (yaw, pitch, and roll) to unit quaternion.

Inputs:

- $\psi \in \mathbb{R}$ - yaw angle (1st rotation, about 3rd axis) [rad]
- $\theta \in \mathbb{R}$ - pitch angle (2nd rotation, about 2nd axis) [rad]
- $\phi \in \mathbb{R}$ - roll angle (3rd rotation, about 1st axis) [rad]

Procedure:

1. Precompute the trigonometric functions.

$$c_1 = \cos\left(\frac{\phi}{2}\right)$$
$$s_1 = \sin\left(\frac{\phi}{2}\right)$$
$$c_2 = \cos\left(\frac{\theta}{2}\right)$$
$$s_2 = \sin\left(\frac{\theta}{2}\right)$$
$$c_3 = \cos\left(\frac{\psi}{2}\right)$$
$$s_3 = \sin\left(\frac{\psi}{2}\right)$$

2. Compute the quaternion.

$$\mathbf{q} = \begin{bmatrix} c_1c_2c_3 + s_1s_2s_3 \\ s_1c_2c_3 - c_1s_2s_3 \\ c_1s_2c_3 + s_1c_2s_3 \\ c_1c_2s_3 - s_1s_2c_3 \end{bmatrix}$$

3. Normalize the quaternion (Algorithm 20).

$$\mathbf{q} = \texttt{quatnormalize}(\mathbf{q})$$

4. Return the result.

return $\mathbf{q}$

Outputs:
- $\mathbf{q} \in \mathbb{H}$ - unit quaternion

Note:
- This algorithm assumes the scalar-first convention for quaternions.
- The scalar component, q_0, is chosen to be positive.

Test Cases:
- See Appendix A.1.2.

4.5.6 Relationship With Axis-Angle Representation

Given a principal rotation vector, $\mathbf{e} = (e_1, e_2, e_3)^T$, and principal angle, Φ, we can define the unit quaternion as [52, p. 23], [39]

$$\mathbf{q} = \begin{bmatrix} \cos\left(\frac{\Phi}{2}\right) \\ \sin\left(\frac{\Phi}{2}\right)\mathbf{e} \end{bmatrix} \begin{bmatrix} \cos\left(\frac{\Phi}{2}\right) \\ e_1 \sin\left(\frac{\Phi}{2}\right) \\ e_2 \sin\left(\frac{\Phi}{2}\right) \\ e_3 \sin\left(\frac{\Phi}{2}\right) \end{bmatrix} \tag{4.55}$$

Recall Convention 26, which defined the sign convention for quaternions such that q_0 is always positive. In this case, if $q_0 < 0$, we must switch the sign on the entire quaternion.

Algorithm 27: `axang2quat`
Axis-angle representation to unit quaternion.

Inputs:
- $\mathbf{e} \in \mathbb{R}^3$ - principal rotation vector
- $\Phi \in \mathbb{R}$ - principal angle $[\text{rad}]$

Procedure:
1. Normalize the principal rotation vector.

$$\mathbf{e} = \frac{\mathbf{e}}{\|\mathbf{e}\|}$$

2. Determine the quaternion.

$$\mathbf{q} = \begin{bmatrix} \cos\left(\frac{\Phi}{2}\right) \\ \sin\left(\frac{\Phi}{2}\right)\mathbf{e} \end{bmatrix}$$

3. Ensure that the scalar component of the quaternion is positive.

if $q_0 < 0$
$\quad \mathbf{q} = -\mathbf{q}$

end

4. Normalize the quaternion (Algorithm 20).

   $\mathbf{q} = \texttt{quatnormalize}(\mathbf{q})$

5. Return the result.

   return $\mathbf{q}$

Outputs:

- $\mathbf{q} \in \mathbb{H}$ - unit quaternion

Note:

- This algorithm normalizes $\mathbf{e}$.
- This algorithm assumes the scalar-first convention for quaternions.
- The scalar component, q_0, is chosen to be positive.

Test Cases:

- See Appendix A.1.2.

To obtain the Euler axis and angle from a unit quaternion, we can just simply solve Eq. (4.55) for $\mathbf{e}$ and Φ. First, solving for Φ,

$$\boxed{\Phi = 2 \arccos q_0} \tag{4.56}$$

Note that Eq. (4.56) will always return a positive Φ when using the quaternion sign convention defined by Convention 26. However, in practice, we need to take the precaution of checking the sign on q_0 and negating the quaternion if needed when assembling the axis-angle representation.

Additionally, due to finite precision arithmetic, q_0 may sometimes be slightly less than -1 or slightly greater than 1, which will cause issues with the inverse cosine function. Thus, we must manually clamp q_0 to have a value between -1 and 1 [10].

Next, we can solve for $\mathbf{e}$ as

$$\mathbf{e} = \frac{1}{\sin(\Phi/2)} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{4.57}$$

However, note that

$$\sin^2\left(\frac{\Phi}{2}\right) + \cos^2\left(\frac{\Phi}{2}\right) = 1 \quad \rightarrow \quad \sin\left(\frac{\Phi}{2}\right) = \sqrt{1 - \cos^2\left(\frac{\Phi}{2}\right)} = \sqrt{1 - q_0^2}$$

Substituting this into Eq. (4.57),

$$\mathbf{e} = \frac{1}{\sqrt{1 - q_0^2}} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{4.58}$$

At this point, we still have an issue; recall that the axis-angle representation has a singularity at $\Phi = 0$. We can see this immediately from Eq. (4.57), and from Eq. (4.58), we can see that this also corresponds to the case where $|q_0| = 1$. Since this singularity corresponds to the case where there is no rotation, we can set the principal rotation vector to

$\mathbf{e} = (1,0,0)^T$ by default, as we did for this same singularity in Section 4.4.1 previously.

$$\boxed{\mathbf{e} = \begin{cases} \dfrac{1}{\sqrt{1-q_0^2}} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}, & |q_0| < 1 \\ (1,0,0)^T, & q_0 = 1 \end{cases}} \tag{4.59}$$

Note that in its implementation, we do not need to compute the coefficient on $\mathbf{e}$ in Eq. (4.59), since we will normalize the principal rotation vector anyways.

Algorithm 28: `quat2axang`
Quaternion to axis-angle representation.

Inputs:

- $\mathbf{q} \in \mathbb{H}$ - quaternion

Procedure:

1. Ensure that the scalar component of the quaternion is positive.

   if $q_0 < 0$
   $\quad \mathbf{q} = -\mathbf{q}$
   end

2. Normalize the quaternion (Algorithm 20).

   $\mathbf{q} = \texttt{quatnormalize}(\mathbf{q})$

3. Principal rotation vector.

   if $|q_0| < 1$
   $\quad \mathbf{e} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}$
   else
   $\quad \mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$
   end

4. Normalize the principal rotation vector.

   $$\mathbf{e} = \frac{\mathbf{e}}{\|\mathbf{e}\|}$$

5. Ensure that $|q_0| \leq 1$.

   $$q_0 = \max\left[\min\left(q_0, 1\right), -1\right]$$

6. Principal angle [rad].

   $$\Phi = 2 \arccos q_0$$

7. Return the result.

   return $\mathbf{e}, \Phi$

Outputs:
- $\mathbf{e} \in \mathbb{R}^3$ - principal rotation vector
- $\Phi \in \mathbb{R}$ - principal angle $[\text{rad}]$

Note:
- This algorithm assumes the scalar-first convention for quaternions.
- This algorithm normalizes the input quaternion.
- $\|\mathbf{e}\| = 1$.
- $\Phi \in [0, \pi]$
- If $\Phi = 0$, then $\mathbf{e}$ is returned as $(1, 0, 0)^T$.

Test Cases:
- See Appendix A.1.2.

4.5.7 Angle Between Unit Quaternions

In general, a quaternion can be viewed as a four-dimensional vector (as discussed in Section 4.5.1), representing a point in a four-dimensional space. However, constraining it to a unit magnitude yields a three dimensional space equivalent to the surface a hypersphere. Alternatively, you can view it as taking the unit vector $(1, 0, 0)^T$, and performing every possible rotation of that unit vector; its tip will trace out the surface of the unit sphere. Thus, a single quaternion can be visualized as a point on the unit sphere resulting from applying a rotation to a unit vector.

The angle between two quaternions is then the angle that subtends the great arc connecting the two corresponding points on the unit sphere [36, 42, 46]. Consider two quaternions $\mathbf{q}_1, \mathbf{q}_2 \in \mathbb{H}$. The angle between them can be computed as [13][8].

$$\boxed{\theta = 4 \arctan2 \left(\|\mathbf{q}_1 - \mathbf{q}_2\|, \|\mathbf{q}_1 + \mathbf{q}_2\| \right)} \tag{4.60}$$

Algorithm 29: `quatang`
Angle between two unit quaternions.

Inputs:
- $\mathbf{q}_1 \in \mathbb{H}$ - unit quaternion
- $\mathbf{q}_2 \in \mathbb{H}$ - unit quaternion

Procedure:

$\theta = 4 \arctan2 \left[\texttt{quatnorm}(\mathbf{q}_1 - \mathbf{q}_2), \texttt{quatnorm}(\mathbf{q}_1 + \mathbf{q}_2) \right]$ (Algorithm 18)

return θ

Outputs:
- $\theta \in \mathbb{R}$ - angle between the unit quaternions $[\text{rad}]$

Note:

[8] Note that [13] actually defines $\theta = 2 \arctan2 \left(\|\mathbf{q}_1 - \mathbf{q}_2\|, \|\mathbf{q}_1 + \mathbf{q}_2\| \right)$. However, this yields exactly *half* of the principal angle from the axis-angle representation of the intermediate rotation between $\mathbf{q}_1$ and $\mathbf{q}_2$.

- This algorithm assumes that $\mathbf{q}_1$ and $\mathbf{q}_2$ are input as unit quaternions, so it does *not* normalize them.

Test Cases:

- See Appendix A.1.3.

The Axis-Angle Approach

This angle can also be found from axis-angle representation of a rotation. Recall from Section 4.5.3 that

$$\mathbf{q}_{A\to C} = \mathbf{q}_{A\to B} \otimes \mathbf{q}_{B\to C}$$

Let's assume we know $\mathbf{q}_{A\to B}$ and $\mathbf{q}_{A\to C}$, and want to find the intermediate rotation $\mathbf{q}_{B\to C}$. Left-multiplying both sides by $\mathbf{q}^*_{A\to B}$, and noting that $\mathbf{q}^*_{A\to B} = \mathbf{q}^{-1}_{A\to B}$ for unit quaternions,

$$\mathbf{q}^{-1}_{A\to B} \otimes \mathbf{q}_{A\to C} = \mathbf{q}^{-1}_{A\to B} \otimes (\mathbf{q}_{A\to B} \otimes \mathbf{q}_{B\to C}) = (\mathbf{q}^{-1}_{A\to B} \otimes \mathbf{q}_{A\to B}) \otimes \mathbf{q}_{B\to C}$$

$$\therefore \mathbf{q}_{B\to C} = \mathbf{q}^{-1}_{A\to B} \otimes \mathbf{q}_{A\to C}$$

In this case, we have the unit quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$, essentially representing two orientations or attitudes. They are linked via an intermediate rotation, $\mathbf{q}$. Therefore, comparing to the example above, we can make the substitutions $\mathbf{q}_1 = \mathbf{q}_{A\to B}$, $\mathbf{q}_2 = \mathbf{A} \to \mathbf{C}$, and $\mathbf{q} = \mathbf{q}_{B\to C}$.

$$\mathbf{q} = \mathbf{q}_1^{-1} \otimes \mathbf{q}_2$$

The angle between $\mathbf{q}_1$ and $\mathbf{q}_2$ is then just the principal angle Φ of the rotation represented by $\mathbf{q}$. We *could* use the following procedure [2]:

1. Calculate $\mathbf{q}_1^*$ using Algorithm 16.

   $$\mathbf{q}_1^* = \texttt{quatconj}(\mathbf{q}_1)$$

2. Calculate $\mathbf{q}$ using Algorithm 17.

   $$\mathbf{q} = \texttt{quatmul}(\mathbf{q}_1^*, \mathbf{q}_q)$$

3. Find θ using Algorithm 28.

   $$[\sim, \theta] = \texttt{quat2axang}(\mathbf{q})$$

However, it is much more efficient to just use Algorithm 29. Nonetheless, we can use this procedure to unit test[9] Algorithm 29.

4.5.8 Spherical Linear Interpolation (SLERP)

Spherical linear interpolation (SLERP) is a method for interpolating between two quaternions. Similar to how linear interpolation interpolates between two points using a line with a constant derivative, SLERP interpolates between two quaternions by moving at a constant speed along a great circle arc on the unit sphere [42, 46].

[9] In fact, this is precisely how we found the aforementioned typo in [13].

Let $t \in [0,1]$ be the **interpolation parameter** representing the fraction of the arc connecting the two quaternions that we want to find a third quaternion via SLERP. Let $\mathbf{q}_1, \mathbf{q}_2 \in \mathbb{H}$ be two quaternions, and let θ smaller of the two angles between them. Then the basic SLERP formula is

$$\boxed{\mathbf{q}(t) = \left(\frac{\sin\left[(1-t)\theta\right]}{\sin\theta}\right)\mathbf{q}_1 + \left(\frac{\sin t\theta}{\sin\theta}\right)\mathbf{q}_2} \tag{4.61}$$

The angle θ can be found using Algorithm 29. However, in practice, we need to incorporate additional local to ensure the shorter of the two arcs is used. This is done in Algorithm 30 below, which is adapted from the procedure outlined in [13].

Algorithm 30: `quatslerp`
Spherical linear interpolation (SLERP) between two unit quaternions.

Inputs:

- $\mathbf{q}_1 \in \mathbb{H}$ - unit quaternion
- $\mathbf{q}_2 \in \mathbb{H}$ - unit quaternion
- t - interpolation parameter between 0 and 1 (inclusive)

Procedure:

1. Angle between the unit quaternions (Algorithm 29) $[\mathrm{rad}]$.

$$\phi = \texttt{quatang}(\mathbf{q}_1, \mathbf{q}_2)$$

2. Define a new angle to ensure we are interpolating along the shorter arc $[\mathrm{rad}]$.

if $\phi > (\pi/2)$
$\quad \theta = \pi - \phi$
else
$\quad \theta = \phi$
end

3. Evaluate the trigonometric functions.

$$a = \sin\theta$$
$$b = \sin(t\theta)$$
$$c = \sin\left[(1-t)\theta\right]$$

4. Auxiliary parameters.

$$d = \frac{c}{a}$$
$$e = \frac{b}{a}$$

5. Perform SLERP along the shorter arc.

if $\phi > (\pi/2)$
$\quad \mathbf{q} = d\mathbf{q}_1 - e\mathbf{q}_2$
else
$\quad \mathbf{q} = d\mathbf{q}_1 + e\mathbf{q}_2$
end

6. Normalize the result (Algorithm 20).

$$\mathbf{q} = \texttt{quatnormalize}(\mathbf{q})$$

7. Return the result.

return $\mathbf{q}$

Outputs:

- $\mathbf{q} \in \mathbb{H}$ - interpolated unit quaternion

Note:

- This algorithm assumes that $\mathbf{q}_1$ and $\mathbf{q}_2$ are input as unit quaternions, so it does *not* normalize them.
- $\mathbf{q}$ is normalized.

Test Cases:

- See Appendix A.1.3.

PART II

Modeling the Environment

5

Measurement of Time

5.1 Time Units

An **epoch** (pronounced "epic") is a moment in time that designates some event, or more broadly designates some reference point in time. We refer to a particular instant in time as a **date**. Thus, an epoch is described by a date. Two commonly used epochs are shown below[1]:

$$\boxed{\text{J2000.0 epoch} = \text{2000 January 1, 12:00:00.000 TT}} \tag{5.1}$$

$$\boxed{\text{GPS epoch} = \text{1980 January 6, 00:00:00.000 UTC/GPS}} \tag{5.2}$$

To be able to determine the epoch of an event, we need both a precise understanding of time and systems of units that are universally agreed upon [55, pp. 174, 183].

5.1.1 Gregorian (Calendar) Dates

The classical way to define an epoch is using a **Gregorian date** in the format YYYY/MM/DD hh:mm:ss. YYYY is the year, MM the month, DD the day, hh the hour, mm the minute, and ss the second. Note that ss can be fractional (i.e. we could have $\mathrm{ss} = 5.12904...$). From a computational perspective, it is convenient to store the Gregorian date in a row vector, $\mathbf{c}$:

$$\boxed{\mathbf{c} = \begin{bmatrix}\mathrm{YYYY} & \mathrm{MM} & \mathrm{DD} & \mathrm{hh} & \mathrm{mm} & \mathrm{ss}\end{bmatrix} \qquad \text{(Gregorian date)}} \tag{5.3}$$

The symbols in the vector above are meant to be used as variables in the various unit conversions. The shorthand for the actual units (including centuries) are defined as follows:

- c – centuries
- y – years
- mo – months
- wk – weeks
- d – days
- h – hours
- m – minutes
- s – seconds

[1] The TT, UTC, and GPS time scales are discussed in Section 5.2.

An example of a date written as a Gregorian date is shown below.

February 1, 2022 16:07:13

Oftentimes, numbers and variables are also subscripted with the units:

$$\text{February 1, 2022 } 16^{\text{h}}07^{\text{m}}13^{\text{s}}$$

Finally, in everyday usage, the Gregorian date is often referred to simply as the **calendar date**. Note that this is a somewhat of a misnomer; since there are different calendars, such as the Julian and Gregorian calendars, "calendar date" can be somewhat ambiguous. However, most of the world now uses the Gregorian calendar, and we use the abbreviation "`cal`" to define the name of functions as well as variables within those functions.

Gregorian (calendar) date $\leftrightarrow$ day of year

In some applications, it is necessary to find the day of the year from the calendar date, and vice-versa. Algorithm 31 for finding the day of year from the Gregorian (calendar) date was developed from the discussion in [55, p. 200].

Algorithm 31: `cal2doy`
Day of year from Gregorian (calendar) date.

Inputs:

- $\mathbf{c} \in \mathbb{R}^{1\times 3}$ - Gregorian (calendar) date $[\text{y}, \text{mo}, \text{d}]$

Procedure:

1. Extract YYYY [y], MM [mo], and DD[d] from $\mathbf{c}$.
2. Vector to store number of days in each month in non-leap years [d].

$$\mathbf{d} = \begin{bmatrix} 31 & 28 & 31 & 30 & 31 & 30 & 31 & 31 & 30 & 31 & 30 & 31 \end{bmatrix}$$

3. Change the number of days in February to 29 if in a leap year [d].

if $(\text{mod}\,(\text{YYYY}, 4) = 0)$
$\quad d_2 = 29$
end

4. Day of year [d].

$$\text{DOY} = \text{DD} + \sum_{i=1}^{\text{MM}-1} d_i$$

5. Return the result.

return DOY

Outputs:

- $\text{DOY} \in \mathbb{Z}$ - day of year [d]

Test Cases:

- See Appendix A.2.1.

To perform the conversion in the opposite direction, we can use Algorithm 32. Note that for the inverse conversion, we require the year as an input, since we need to know if it is a leap year or not.

Algorithm 32: `doy2cal`
Day of year from Gregorian (calendar) date.

Inputs:
- $\mathrm{YYYY} \in \mathbb{Z}$ - year $[\mathrm{y}]$
- $\mathrm{DOY} \in \mathbb{Z}$ - day of year $[\mathrm{d}]$

Procedure:
1. Vector to store number of days in each month in non-leap years $[\mathrm{d}]$.

$$\mathbf{d} = \begin{bmatrix} 31 & 28 & 31 & 30 & 31 & 30 & 31 & 31 & 30 & 31 & 30 & 31 \end{bmatrix}$$

2. Change the number of days in February to 29 if in a leap year $[\mathrm{d}]$.

if $(\mathrm{mod}\,(\mathrm{YYYY}, 4) = 0)$
$\quad d_2 = 29$
end

3. Determine the month $[\mathrm{mo}]$.

$\mathrm{MM} = 1$
while $\left(\sum_{i=1}^{\mathrm{MM}} d_i < \mathrm{DOY}\right)$
$\quad \mathrm{MM} = \mathrm{MM} + 1$
end

4. Determine the day of the month $[\mathrm{d}]$.

if $(\mathrm{MM} = 1)$
$\quad \mathrm{DD} = \mathrm{DOY}$
else
$\quad \mathrm{DD} = \mathrm{DOY} - \sum_{i=1}^{\mathrm{MM}-1} d_i$
end

5. Define the $\mathbf{c}$ vector storing the Gregorian (calendar) date.

$$\mathbf{c} = \begin{bmatrix} \mathrm{YYYY} & \mathrm{MM} & \mathrm{DD} & 0 & 0 & 0 \end{bmatrix}$$

6. Return the result.

return $\mathbf{c}$

Outputs:
- $\mathbf{c} \in \mathbb{R}^{1\times 6}$ - Gregorian (calendar) date $[\mathrm{y}, \mathrm{mo}, \mathrm{d}, \mathrm{h}, \mathrm{m}, \mathrm{s}]$

Note:
- This implementation is only valid between the years 1901 and 2099.

Test Cases:
- See Appendix A.2.1.

5.1.2 Julian and Modified Julian Dates

Julian date

The **Julian date**, JD, is the number of days from the epoch January 1, 4713 B.C., 12:00 [30, p. 319].

Modified Julian date

The **modified Julian date**, MJD, is defined as [30, p. 319]

$$\boxed{\mathrm{MJD} = \mathrm{JD} - 2400000.5} \tag{5.4}$$

If we know Julian date, then we can use Algorithm 34 to calculate the modified Julian date.

Algorithm 33: `jd2mjd`
Modified Julian date from Julian date.

Inputs:
- $\mathrm{JD} \in \mathbb{R}$ - Julian date [d]

Procedure:

$\mathrm{MJD} = \mathrm{JD} - 2400000.5$
return MJD

Outputs:
- $\mathrm{MJD} \in \mathbb{R}$ - modified Julian date [d]

Test Cases:
- See Appendix A.2.1.

Rearranging Eq. (5.4),

$$\boxed{\mathrm{JD} = \mathrm{MJD} + 2400000.5} \tag{5.5}$$

We formalize this as Algorithm 34.

Algorithm 34: `mjd2jd`
Julian date from modified Julian date.

Inputs:
- $\mathrm{MJD} \in \mathbb{R}$ - modified Julian date [d]

Procedure:

$\mathrm{JD} = \mathrm{MJD} + 2400000.5$

return JD

Outputs:

- $\mathrm{JD} \in \mathbb{R}$ - Julian date [d]

Test Cases:

- See Appendix A.2.1.

Julian centuries since J2000.0

The variable T is used to denote **Julian centuries since J2000.0**. It is defined as

$$\boxed{T = \frac{\mathrm{JD} - 2451545}{36525}} \tag{5.6}$$

Note that T and JD are in the same time scale (see Section 5.2 for more information on time scales). For example, if we are given the Julian date of TT, $\mathrm{JD}_{\mathrm{TT}}$, then Eq. (5.6) will give us T_{TT} [55, p. 184]. Formalizing this equation as Algorithm 35,

Algorithm 35: `jd2t`
Julian centuries since J2000.0 from Julian date.

Inputs:

- $\mathrm{JD} \in \mathbb{R}$ - Julian date [d]

Procedure:

$T = \dfrac{\mathrm{JD} - 2451545}{36525}$

return T

Outputs:

- $T \in \mathbb{R}$ - Julian centuries since J2000.0 [c]

Test Cases:

- See Appendix A.2.1.

Note that if we substitute Eq. (5.5) into Eq. (5.6), we can also write T in terms of MJD as

$$T = \frac{(\mathrm{MJD} + 2400000.5) - 2451545}{36525} \quad \rightarrow \quad \boxed{T = \frac{\mathrm{MJD} - 51544.5}{36525}} \tag{5.7}$$

Formalizing Eq. (5.7) as Algorithm 36,

Algorithm 36: `mjd2t`
Julian centuries since J2000.0 from Julian date.

Inputs:
- $\mathrm{MJD} \in \mathbb{R}$ - modified Julian date $[\mathrm{d}]$

Procedure:

$$T = \frac{\mathrm{MJD} - 51544.5}{36525}$$

return T

Outputs:
- $T \in \mathbb{R}$ - Julian centuries since J2000.0 $[\mathrm{c}]$

Test Cases:
- See Appendix A.2.1.

Finally, it is *essential* to note that T_{GPS}, T_{TAI}, T_{UT1}, T_{UTC} do ***not*** represent **exact** centuries since the J2000.0 epoch. This is because the J2000.0 epoch is defined as 2000 January 1, 12:00:00.000 TT, which corresponds to a modified Julian date of TT of $(\mathrm{MJD}_{\mathrm{TT}})_{\mathrm{J2000.0}} = 51544.5$. However, at the J2000.0 epoch,

$$(\mathrm{MJD}_{\mathrm{GPS}})_{\mathrm{J2000.0}} \neq (\mathrm{MJD}_{\mathrm{TAI}})_{\mathrm{J2000.0}} \neq (\mathrm{MJD}_{\mathrm{UT1}})_{\mathrm{J2000.0}} \neq (\mathrm{MJD}_{\mathrm{UTC}})_{\mathrm{J2000.0}} \neq 51544.5$$

Nonetheless, the Julian centuries since J2000.0 for *any* of the time scales is defined using Eqs. (5.6) and (5.7); any equations/algorithm that make use of T are already written accordingly [32, pp. 45, 52], [55, p. 184].

Gregorian (calendar) date $\leftrightarrow$ modified Julian date

Algorithm 37 below (adapted from Appendix A.1.1 in [30, pp. 321–322] and) converts a Gregorian (calendar) date to a modified Julian date. Note that the aforementioned references account for dates before 1582 October 15[2]. However, to simplify our function, we assume the date is on or after October 15, 1582, which is true for virtually all practical applications today.

Algorithm 37: `cal2mjd`
Modified Julian date from Gregorian (calendar) date.

Inputs:
- $\mathbf{c} \in \mathbb{R}^{1\times 6}$ - Gregorian (calendar) date $[\mathrm{y}, \mathrm{mo}, \mathrm{d}, \mathrm{h}, \mathrm{m}, \mathrm{s}]$

Procedure:
1. Extract YYYY, MM, DD, hh, mm, and ss from $\mathbf{c}$.
2. Throw an error if the date is before 1582 October 15.

[2] Due to the Gregorian calendar reform, there are no dates between 1582 October 4 and 1582 October 15 [19], [31, p. 15]; the algorithms presented [31, pp. 14–15] and [30, pp. 321–322] account for this. Note that there appears to be a slight error Eq. (A.5) in [30, p. 321]; it uses October 10, 1582, which is a date that does not exist.

if $(\mathrm{YYYY} \leq 1582)$ **and** $[(\mathrm{MM} < 10)$ **or** $((\mathrm{MM} = 10)$ **and** $(\mathrm{DD} < 15))]$
  `error`("This function is only valid for dates after 1582 October 14.")
end

3. Convert time to fraction of a day (Algorithm 39) [d].

   $f_{\mathrm{DD}} = \texttt{hms2f}(\mathrm{hh}, \mathrm{mm}, \mathrm{ss})$

4. Append time to day [d].

   $\mathrm{DD} = \mathrm{DD} + f_{\mathrm{DD}}$

5. Handle leap years.

   if $\mathrm{MM} \leq 2$
     $y = \mathrm{YYYY} - 1$
     $m = \mathrm{MM} + 12$
   else
     $y = \mathrm{YYYY}$
     $m = \mathrm{MM}$
   end

6. Handle leap days.

   $$B = \left\lfloor \frac{y}{400} \right\rfloor - \left\lfloor \frac{y}{100} \right\rfloor + \left\lfloor \frac{y}{4} \right\rfloor$$

7. Modified Julian date.

   $$\mathrm{MJD} = 365y - 679004 + B + \lfloor 30.6001(m+1) \rfloor + \mathrm{DD}$$

8. Return the result.

   return MJD

Outputs:

- $\mathrm{MJD} \in \mathbb{R}$ - modified Julian date [d]

Note:

- This implementation is only valid for dates after 1582 October 14.

Test Cases:

- See Appendix A.2.1.

To perform the conversion in the opposite direction, we can use Algorithm 38, which is adapted from Appendix A.1.2 in [30, pp. 322–323]. Similar to how Algorithm 37 does not support dates before 1582 October 15, Algorithm 38 does not support modified Julian dates before -100840 (corresponding to 1582 October 15). Note that the procedure is also slightly modified to return the fraction of the day in hours, minutes, and seconds as well.

Algorithm 38: `mjd2cal`
Gregorian (calendar) date from modified Julian date.

Inputs:

- $\mathrm{MJD} \in \mathbb{R}$ - modified Julian date $[\mathrm{d}]$

Procedure:

1. Integer Julian Day (Julian date at noon) $[\mathrm{JD}]$.

$$a = \lfloor \mathrm{MJD} \rfloor + 2400001$$

2. Auxiliary quantities.

$$b = \left\lfloor \frac{a - 1867216.25}{36524.25} \right\rfloor$$

$$c = a + b - \left\lfloor \frac{b}{4} \right\rfloor + 1525$$

$$d = \left\lfloor \frac{c - 121.1}{365.25} \right\rfloor$$

$$e = \lfloor 365.25d \rfloor$$

$$f = \left\lfloor \frac{c - e}{30.6001} \right\rfloor$$

3. Day of month $[\mathrm{d}]$.

$$\mathrm{DD} = c - e - \lfloor 30.6001f \rfloor + f_{\mathrm{DD}}$$

4. Month of year $[\mathrm{mo}]$.

$$\mathrm{MM} = f - 1 - 12 \left\lfloor \frac{f}{14} \right\rfloor$$

5. Year $[\mathrm{y}]$.

$$\mathrm{YYYY} = d - 4715 - \left\lfloor \frac{7 + \mathrm{MM}}{10} \right\rfloor$$

6. Fraction of day (Algorithm 41) $[\mathrm{d}]$.

$$f_{\mathrm{DD}} = \texttt{mjd2f}(\mathrm{MJD})$$

7. Hours, minutes, and seconds from fraction of day (Algorithm 40) $[\mathrm{h}, \mathrm{m}, \mathrm{s}]$.

$$[\mathrm{hh}, \mathrm{mm}, \mathrm{ss}] = \texttt{f2hms}(f_{\mathrm{DD}})$$

8. Define the $\mathbf{c}$ vector storing the Gregorian (calendar) date $[\mathrm{y}, \mathrm{mo}, \mathrm{d}, \mathrm{h}, \mathrm{m}, \mathrm{s}]$.

$$\mathbf{c} = \begin{bmatrix} \mathrm{YYYY} & \mathrm{MM} & \mathrm{DD} & \mathrm{hh} & \mathrm{mm} & \mathrm{ss} \end{bmatrix}$$

9. Return the result.

return c

Outputs:

- $\mathbf{c} \in \mathbb{R}^{1\times 6}$ - Gregorian (calendar) date $[\mathrm{y}, \mathrm{mo}, \mathrm{d}, \mathrm{h}, \mathrm{m}, \mathrm{s}]$

Note:

- This implementation is only valid for dates after 1582 October 14.

Test Cases:

- See Appendix A.2.1.

5.1.3 Time Measurement Within a Day

The time within a single day is typically tracked with hours, minutes, and seconds.

Fraction of a day ↔ hours, minutes, seconds

Let f_{DD} represent the fraction of a day. To obtain the fraction of the day from the elapsed hours (hh), minutes (mm), and seconds (ss) since the beginning of the day,

$$f_{\mathrm{DD}} = \left[\frac{\mathrm{hh}}{24}\right] + \left[\frac{\mathrm{mm}}{(24\ \mathrm{h/d})(60\ \mathrm{m/h})}\right] + \left[\frac{\mathrm{ss}}{(24\ \mathrm{h/d})(60\ \mathrm{m/h})(60\ \mathrm{s/m})}\right]$$

$$\boxed{f_{\mathrm{DD}} = \left(\frac{\mathrm{hh}}{24}\right) + \left(\frac{\mathrm{mm}}{1440}\right) + \left(\frac{\mathrm{ss}}{86400}\right)} \tag{5.8}$$

We formalize this as Algorithm 39.

Algorithm 39: `hms2f`
Fraction of day from hours, minutes, and seconds.

Inputs:
- $\mathrm{hh} \in \mathbb{Z}$ - hours [h]
- $\mathrm{mm} \in \mathbb{Z}$ - minutes [m]
- $\mathrm{ss} \in \mathbb{R}$ - seconds [s]

Procedure:

$$f_{\mathrm{DD}} = \left(\frac{\mathrm{hh}}{24}\right) + \left(\frac{\mathrm{mm}}{1440}\right) + \left(\frac{\mathrm{ss}}{86400}\right)$$

return f_{DD}

Outputs:
- $f_{\mathrm{DD}} \in \mathbb{R}$ - fraction of day [d]

Test Cases:
- See Appendix A.2.1.

To perform the conversion in the opposite direction, we can use Algorithm 40 [55, p. 199].

Algorithm 40: `f2hms`
Hours, minutes, and seconds from fraction of day.

Inputs:
- $f_{\mathrm{DD}} \in \mathbb{R}$ - fraction of day [d]

Procedure:
1. Hours.

$$\mathrm{hh} = \lfloor 24 f_{\mathrm{DD}} \rfloor$$

2. Minutes.

$$\mathrm{mm} = \lfloor 60(24 f_{\mathrm{DD}} - \mathrm{hh}) \rfloor$$

3. Seconds.

$$\mathrm{ss} = 3600 \left(24 f_{\mathrm{DD}} - \mathrm{hh} - \frac{\mathrm{mm}}{60} \right)$$

4. Return the result.

return $\mathrm{hh}, \mathrm{mm}, \mathrm{ss}$

Outputs:

- $\mathrm{hh} \in \mathbb{Z}$ - hours [h]
- $\mathrm{mm} \in \mathbb{Z}$ - minutes [m]
- $\mathrm{ss} \in \mathbb{R}$ - seconds [s]

Test Cases:

- See Appendix A.2.1.

Modified Julian date $\rightarrow$ fraction of a day

Given a positive modified Julian date, $\mathrm{MJD} > 0$, we can easily determine the fraction of the day as

$$f_{\mathrm{DD}} = \mathrm{MJD} - \lfloor \mathrm{MJD} \rfloor , \quad \mathrm{MJD} > 0$$

Note that this also works for negative modified Julian dates. We formalize this as Algorithm 41.

Algorithm 41: `mjd2f`
Fraction of day from modified Julian date.

Inputs:

- $\mathrm{MJD} \in \mathbb{R}$ - modified Julian date [d]

Procedure:

$f_{\mathrm{DD}} = \mathrm{MJD} - \lfloor \mathrm{MJD} \rfloor$
return f_{DD}

Outputs:

- $f_{\mathrm{DD}} \in \mathbb{R}$ - fraction of day [d]

Test Cases:

- See Appendix A.2.1.

5.2 Time Scales

Measuring Time

There are five main ways for measuring time that we consider for astrodynamic purposes:

1. **Solar Time:** Defined by successive transits of the Sun over a particular meridian (i.e. line of longitude). Specifically, one solar day is the time required for an observer on the Earth to revolve once *and* observe the Sun in the same location [55, p. 177].
2. **Sidereal Time:** Defined by successive transits of the stars over a particular meridian (i.e. line of longitude) [55, p. 176]. Sidereal time is similar to solar time, but it "baselines" time against the stars and not against the Sun.
   - Since an Earth-fixed frame is effectively motionless with respect to distant stars, the Earth will rotate through exactly 360° in a sidereal day.
   - In contrast, due to the orbital motion of the Earth around the Sun, the Earth will move approximately 1° in its orbit around the Sun over the course of the day (since there just over 360 days in a year), so the Earth will rotate through approximately 361° in a solar day. Thus, solar time and sidereal time increase at different rates.
3. **Universal Time:** Universal time was originally meant to define a standard for solar time by measuring time via observing successive transits of a *fictitious mean Sun* over the Greenwich meridian [54], [55, pp. 177–178] (i.e. universal time = mean solar time at Greenwich [55, p. 178]). However, due to the difficulty in precisely measuring the position of the Sun, the original realization of Universal time (called UT0) used measurements of the positions of stars. Nowadays, we use UT1 and UTC, which are covered further below.
4. **Atomic Time:** Both solar and sidereal time do not increase at a constant rate due to many factors (the Earth wobbles, the Earth's orbit is slightly elliptical, etc.). Atomic time is baselined with respect to the specific quantum transition of electrons in a cesium-133 atom. Since this transition is extremely deterministic, atomic time essentially represents a way to measure time in a constant fashion [55, p. 190].
   - Note that the length of a second was also chosen to be based loosely on solar time, and not sidereal time.
5. **Dynamical Time:** All the times above are traditionally defined as being measured on the Earth. However, due to our motion with respect to "true" inertial space and the gravity we experience, our measurement of time is fundamentally different to any other point in the universe due to general relativity. This is consequential when generating planetary ephemerides, since our experience of time is different than elsewhere in the solar system. Dynamical time is meant to measure time in such a way that it can be an independent variable of the equations of celestial mechanics. There have been many definitions of dynamical time in the previous decades, many of which were based on measurements of the SI second in different reference frames. However, this would mean that the rate of dynamical time differed from the SI second on Earth. The current realization of dynamical time uses a rate that matches the SI second on Earth [16], [55, pp. 190–193].

Time Scales

There are **five** main time scales we use in most practical applications [55, pp. 179–181, 190]:

1. **UT1 (Universal Time 1)**: Universal time that accounts for the polar motion of the Earth. Since the polar motion of the Earth is not uniform, UT1 does not increase in a constant manner [54], [55, p. 179].
2. **UTC (Coordinate Universal Time)**: UTC is designed to follow UT1, but increases constantly and is calculated using atomic clocks. Since UT1 varies irregularly due to variations in the Earth's rotation (as mentioned above), leap seconds are used to keep UTC within $\pm 0.9^{\text{s}}$ of UT1 [54], [55, pp. 180–181].
3. **GPS:** GPS time increases constantly at the same rate as UTC time[3], but it does *not* use leap seconds to stay within $\pm 0.9^{\text{s}}$ of UT1. Therefore, it differs from UTC by the number of leap seconds [55, p. 181].

[3] To within 1 microsecond per day [55, p. 181].

4. **TAI (International Atomic Time):** TAI is defined independent of the average rotation of the Earth. It is based on counting the cycles of a high-frequency electrical circuit maintained in resonance with a cesium-133 atomic transition, and serves as the source of truth for defining a constant, linearly increasing definition of time, which is the **SI second**. Thus, UTC and GPS are based on offsets to TAI. UTC is offset from TAI by a certain number of leap seconds (since UTC is designed to stay within $\pm 0.9^{\text{s}}$ of UT1), and GPS is always offset from TAI by 19 seconds [55, pp. 177, 190].
5. **TT (Terrestrial Time):** TT is the realization of terrestrial dynamical time (i.e. dynamical time for the Earth). Although atomic time (realized by TAI, and used to define GPS and UTC time) is defined to realize atomic time, we actually use the TAI time scale to define dynamical time in the form of TT time[4]. Due to its historical definition, TT time is offset from TAI time by 32.184 seconds [55, pp. 190–191].

One time scale that is important to know about, but is generally *not* used in practice (unless the extra fidelity is needed), is **barycentric dynamical time (TDB)**. It is rigorously defined as the "independent variable of [the] equations of motion with respect to the barycenter of the solar system" [55, p. 191]; thus, it is meant to be used with the planetary ephemerides provided by JPL. However, recall the discussion earlier in this section about how earlier realizations of dynamical time had different rates to "Earth" time. In contrast, TDB is specifically defined so that a second in barycentric dynamical time matches a second as measured on Earth; this allows TDB to follow TT closely, to a maximum difference of 1.7 ms. Thus, in practice we use terrestrial time (TT) instead of TDB – for most models using TDB, the effect of the difference between TT and TDB is largely insignificant [32, p. 61], [55, p. 193].

5.2.1 Converting Between Time Scales

The relationships between the various time scales can be summarized as [55, p. 190]

$$\begin{aligned} \text{UTC} &= \text{UT1} - \Delta\text{UT1} \\ \text{TAI} &= \text{UTC} + \Delta\text{AT} \\ \text{TT} &= \text{TAI} + 32.184^{\text{s}} \\ \text{TAI} &= \text{GPS} + 19.0^{\text{s}} \end{aligned}$$

A discussion of how to obtain ΔUT1 and ΔAT is included at the end of this section.

In general, we typically have UTC and UT1 as modified Julian dates, and ΔUT1 in seconds. Thus,

$$\boxed{\text{MJD}_{\text{UTC}} = \text{MJD}_{\text{UT1}} - \left[\frac{(\Delta\text{UT1})^{\text{s}}}{86400}\right] \quad \leftrightarrow \quad \text{MJD}_{\text{UT1}} = \text{MJD}_{\text{UTC}} + \left[\frac{(\Delta\text{UT1})^{\text{s}}}{86400}\right]} \tag{5.9}$$

The conversion factor (1/86400) comes from the fact that there are $(24)(3600) = 86400$ seconds in a day. Similarly, ΔAT is also typically expressed in seconds. Thus, we have

$$\boxed{\text{MJD}_{\text{TAI}} = \text{MJD}_{\text{UTC}} + \left[\frac{(\Delta\text{AT})^{\text{s}}}{86400}\right] \quad \leftrightarrow \quad \text{MJD}_{\text{UTC}} = \text{MJD}_{\text{TAI}} - \left[\frac{(\Delta\text{AT})^{\text{s}}}{86400}\right]} \tag{5.10}$$

The remaining two conversions can be written as

$$\boxed{\text{MJD}_{\text{TT}} = \text{MJD}_{\text{TAI}} + \left(\frac{32.184}{86400}\right) \quad \leftrightarrow \quad \text{MJD}_{\text{TAI}} = \text{MJD}_{\text{TT}} - \left(\frac{32.184}{86400}\right)} \tag{5.11}$$

$$\boxed{\text{MJD}_{\text{TAI}} = \text{MJD}_{\text{GPS}} + \left(\frac{19}{86400}\right) \quad \leftrightarrow \quad \text{MJD}_{\text{GPS}} = \text{MJD}_{\text{TAI}} - \left(\frac{19}{86400}\right)} \tag{5.12}$$

The time systems we use in day-to-day life are based on UTC. Therefore, we typically know an epoch in UTC and wish to convert to the other time scales. This can be achieved with the following basic procedure:

[4] This is because TAI is measured on Earth, so it is essentially an approximation of the true dynamical time of Earth.

1. Convert UTC to UT1 using Eq. (5.9).
2. Convert UTC to TAI using Eq. (5.10).
3. Convert TAI to TT using Eq. (5.11).
4. Convert TAI to GPS using Eq. (5.12).

For convenience, we also include algorithms for these conversions below.

UTC ↔ UT1

Algorithm 42: `utc2ut1`
UT1 from UTC.

Inputs:
- $\mathrm{MJD}_{\mathrm{UTC}} \in \mathbb{R}$ - UTC (Universal Coordinated Time) [MJD]
- $(\Delta\mathrm{UT1})^{\mathrm{s}} \in \mathbb{R}$ - difference between UT1 and UTC ($\Delta\mathrm{UT1} = \mathrm{UT1} - \mathrm{UTC}$) [s]

Procedure:

$$\mathrm{MJD}_{\mathrm{UT1}} = \mathrm{MJD}_{\mathrm{UTC}} + \left[\frac{(\Delta\mathrm{UT1})^{\mathrm{s}}}{86400}\right]$$

Outputs:
- $\mathrm{MJD}_{\mathrm{UT1}} \in \mathbb{R}$ - UT1 (Universal Time 1) [MJD]

Test Cases:
- See Appendix A.2.2.

Algorithm 43: `ut12utc`
UTC from UT1.

Inputs:
- $\mathrm{MJD}_{\mathrm{UT1}} \in \mathbb{R}$ - UT1 (Universal Time 1) [MJD]
- $(\Delta\mathrm{UT1})^{\mathrm{s}} \in \mathbb{R}$ - difference between UT1 and UTC ($\Delta\mathrm{UT1} = \mathrm{UT1} - \mathrm{UTC}$) [s]

Procedure:

$$\mathrm{MJD}_{\mathrm{UTC}} = \mathrm{MJD}_{\mathrm{UT1}} - \left[\frac{(\Delta\mathrm{UT1})^{\mathrm{s}}}{86400}\right]$$

Outputs:
- $\mathrm{MJD}_{\mathrm{UTC}} \in \mathbb{R}$ - UTC (Universal Coordinated Time) [MJD]

Test Cases:
- See Appendix A.2.2.

UTC ↔ TAI

Algorithm 44: `utc2tai`
TAI from UTC.

Inputs:
- $\mathrm{MJD}_{\mathrm{UTC}} \in \mathbb{R}$ - UTC (Universal Coordinated Time) [MJD]
- $(\Delta\mathrm{AT})^{\mathrm{s}} \in \mathbb{Z}$ - difference between TAI and UTC ($\Delta\mathrm{AT} = \mathrm{TAI} - \mathrm{UTC}$) [s]

Procedure:

$$\mathrm{MJD}_{\mathrm{TAI}} = \mathrm{MJD}_{\mathrm{UTC}} + \left[\frac{(\Delta\mathrm{AT})^{\mathrm{s}}}{86400}\right]$$

Outputs:
- $\mathrm{MJD}_{\mathrm{TAI}} \in \mathbb{R}$ - TAI (International Atomic Time) [MJD]

Test Cases:
- See Appendix A.2.2.

Algorithm 45: `tai2utc`
UTC from TAI.

Inputs:
- $\mathrm{MJD}_{\mathrm{TAI}} \in \mathbb{R}$ - TAI (International Atomic Time) [MJD]
- $(\Delta\mathrm{AT})^{\mathrm{s}} \in \mathbb{Z}$ - difference between TAI and UTC ($\Delta\mathrm{AT} = \mathrm{TAI} - \mathrm{UTC}$) [s]

Procedure:

$$\mathrm{MJD}_{\mathrm{UTC}} = \mathrm{MJD}_{\mathrm{TAI}} - \left[\frac{(\Delta\mathrm{AT})^{\mathrm{s}}}{86400}\right]$$

Outputs:
- $\mathrm{MJD}_{\mathrm{UTC}} \in \mathbb{R}$ - UTC (Universal Coordinated Time) [MJD]

Test Cases:
- See Appendix A.2.2.

TAI $\leftrightarrow$ TT

Algorithm 46: `tai2tt`
TT from TAI.

Inputs:
- $\mathrm{MJD}_{\mathrm{TAI}} \in \mathbb{R}$ - TAI (International Atomic Time) [MJD]

Procedure:

$$\mathrm{MJD}_{\mathrm{TT}} = \mathrm{MJD}_{\mathrm{TAI}} + \left(\frac{32.184}{86400}\right)$$

Outputs:

- $\mathrm{MJD}_{\mathrm{TT}} \in \mathbb{R}$ - TT (Terrestrial Time) [MJD]

Test Cases:

- See Appendix A.2.2.

Algorithm 47: `tt2tai`
TAI from TT.

Inputs:

- $\mathrm{MJD}_{\mathrm{TT}} \in \mathbb{R}$ - TT (Terrestrial Time) [MJD]

Procedure:

$$\mathrm{MJD}_{\mathrm{TAI}} = \mathrm{MJD}_{\mathrm{TT}} - \left(\frac{32.184}{86400}\right)$$

Outputs:

- $\mathrm{MJD}_{\mathrm{TAI}} \in \mathbb{R}$ - TAI (International Atomic Time) [MJD]

Test Cases:

- See Appendix A.2.2.

TAI $\leftrightarrow$ GPS

Algorithm 48: `tai2gps`
GPS time from TAI.

Inputs:

- $\mathrm{MJD}_{\mathrm{TAI}} \in \mathbb{R}$ - TAI (International Atomic Time) [MJD]

Procedure:

$$\mathrm{MJD}_{\mathrm{GPS}} = \mathrm{MJD}_{\mathrm{TAI}} - \left(\frac{19}{86400}\right)$$

Outputs:

- $\mathrm{MJD}_{\mathrm{GPS}} \in \mathbb{R}$ - GPS time [MJD]

Test Cases:

- See Appendix A.2.2.

Algorithm 49: `gps2tai`
TAI time from GPS.

Inputs:
- $\mathrm{MJD}_{\mathrm{GPS}} \in \mathbb{R}$ - GPS time [MJD]

Procedure:

$$\mathrm{MJD}_{\mathrm{TAI}} = \mathrm{MJD}_{\mathrm{GPS}} + \left(\frac{19}{86400}\right)$$

Outputs:
- $\mathrm{MJD}_{\mathrm{TAI}} \in \mathbb{R}$ - TAI (International Atomic Time) [MJD]

Test Cases:
- See Appendix A.2.2.

GPS $\leftrightarrow$ weeks and seconds

Recall from Section 5.1 that the GPS epoch is defined as

$$\text{GPS epoch} = \text{1980 January 6, 00:00:00.000 UTC/GPS}$$

At this epoch, the GPS time was equal to UTC (now, they differ by an integer number of leap seconds). The modified Julian date of the GPS epoch is

$$(\mathrm{MJD}_{\mathrm{GPS}})_{\mathrm{GPS}} = 44244$$

The integer number of weeks since the GPS epoch is then

$$\boxed{\mathrm{wk} = \left\lfloor \frac{\mathrm{MJD}_{\mathrm{GPS}} - 44244}{7} \right\rfloor} \tag{5.13}$$

The remaining fraction of a week is then

$$f_{\mathrm{wk}} = \left(\frac{\mathrm{MJD}_{\mathrm{GPS}} - 44244}{7}\right) - \mathrm{wk}$$

The number of seconds in a week is

$$\left(\frac{3600\ \mathrm{s}}{\mathrm{h}}\right)\left(\frac{24\ \mathrm{h}}{\mathrm{d}}\right)\left(\frac{7\ \mathrm{d}}{\mathrm{wk}}\right) = 604800\ \mathrm{s/wk}$$

The remaining fraction of a week in seconds is then

$$\mathrm{s} = 604800 f_{\mathrm{wk}} = 604800\left[\left(\frac{\mathrm{MJD}_{\mathrm{GPS}} - 44244}{7}\right) - \mathrm{wk}\right]$$

$$\boxed{\mathrm{s} = 86400\,(\mathrm{MJD}_{\mathrm{GPS}} - 44244) - 604800(\mathrm{wk})} \tag{5.14}$$

Algorithm 50: `gps2wks`
GPS week and seconds from GPS time.

Inputs:
- $\mathrm{MJD}_{\mathrm{GPS}}$ - GPS time [MJD]

Procedure:
1. GPS weeks.

$$\mathrm{wk} = \left\lfloor \frac{\mathrm{MJD}_{\mathrm{GPS}} - 44244}{7} \right\rfloor$$

2. GPS seconds.

$$\mathrm{s} = \left(\frac{\mathrm{MJD}_{\mathrm{GPS}} - 44244}{4233600} \right) - \left(\frac{\mathrm{wk}}{604800} \right)$$

Outputs:
- $\mathrm{wk} \in \mathbb{Z}$ - GPS week [wk]
- $\mathrm{s} \in \mathbb{R}$ - GPS seconds [s]

Test Cases:
- See Appendix A.2.2.

The conversion in the opposite direction can be performed in a single line as [30, pp. 324–325]

$$\boxed{\mathrm{MJD}_{\mathrm{GPS}} = (7)(\mathrm{wk}) + \left(\frac{\mathrm{s}}{86400} \right) + 44244} \tag{5.15}$$

Algorithm 51: `wks2gps`
GPS time from GPS week and seconds.

Inputs:
- $\mathrm{wk} \in \mathbb{Z}$ - GPS week [wk]
- $\mathrm{s} \in \mathbb{R}$ - GPS seconds [s]

Procedure:

$$\mathrm{MJD}_{\mathrm{GPS}} = (7)(\mathrm{wk}) + \left(\frac{\mathrm{s}}{86400} \right) + 44244$$

return $\mathrm{MJD}_{\mathrm{GPS}}$

Outputs:
- $\mathrm{MJD}_{\mathrm{GPS}}$ - GPS time [MJD]

Test Cases:
- See Appendix A.2.2.

5.2.2 Obtaining the Offsets

Both offsets are tabulated as a function of the modified Julian date. We assume this can be the modified Julian date of any time scale.

Offset Between UT1 and UTC

Recall from Section 5.2.1 that UT1 and UTC time are related by the offset $\Delta\mathrm{UT1}$. This offset is reported in various IERS Bulletins:

- Bulletin A: https://datacenter.iers.org/data/latestVersion/6_BULLETIN_A_V2013_016.txt
  - reported in seconds
  - rounded to nearest $(10^{-6})^{\mathrm{s}}$
  - one less digit reported than Bulletin B, but includes predicted values
- Bulletin B: https://datacenter.iers.org/data/latestVersion/207_BULLETIN_B207.txts
  - reported in milliseconds
  - rounded to nearest $(10^{-7})^{\mathrm{s}}$
  - most accurate
- Bulletin D: https://datacenter.iers.org/data/latestVersion/17_BULLETIN_D17.txt
  - reported in seconds
  - rounded to nearest $(10^{-1})^{\mathrm{s}}$
  - least accurate

In practice, it is most useful to download the Earth orientation data files published by IERS. We will need the Earth orientation data files anyway, and the data includes the finalized historical $\Delta\mathrm{UT1}$ data reported in seconds and rounded to the nearest $(10^{-7})^{\mathrm{s}}$ (i.e. the values reported in Bulletin A), as well as the predicted values from Bulletin B.

- HTML version: https://datacenter.iers.org/data/html/finals2000A.data.html
- CSV version: https://datacenter.iers.org/data/csv/finals2000A.data.csv
- All versions: https://www.iers.org/IERS/EN/DataProducts/EarthOrientationData/eop.html

To see all versions, navigate to the third link and click on the "`version meta-data`" link next to "`finals.data (IAU2000)`" in the "`Standard EOP data files`" section of the "`Rapid data and predictions`" table. However, it is usually easiest to just use the `.csv` file provided by the second link.

Algorithm 52: `get_dut1`

Obtains the difference between UT1 and UTC ($\Delta\mathrm{UT1} = \mathrm{UT1} - \mathrm{UTC}$).

Inputs:

- $\mathrm{MJD} \in \mathbb{R}$ - modified Julian date of any time scale [MJD]

Procedure:

1. Load the $\Delta\mathrm{UT1}$ vs. MJD data from one of the IERS Earth orientation parameters

file discussed previously in this section. Store the ΔUT1 values in the vector $\mathbf{y} = (y_1, ..., y_N)^T$ and the MJD data in the vector $\mathbf{x} = (y_1, ..., y_N)^T$.

2. Find the number of data points, N, given that $\mathbf{x} \in \mathbb{R}^n$.

$$N = \texttt{length}(\mathbf{x})$$

3. Edge case: the specified date is after the last available date in the data set.

if $\text{MJD} \geq x_N$
$\quad \Delta\text{UT1} = y_u$

4. Base case: the specified date is either (a) before the first available date in the data set or (b) contained within the available dates in the data set.

else

(a) Find the lower bound of the interval of $\mathbf{x}$ that contains MJD. This can be done using the `find_interval` algorithm from [22]. Note that this algorithm will automatically return the first interval of $\mathbf{x}$ if $\text{MJD} < x_1$.

$$[l, \sim] = \texttt{find\_interval}(\mathbf{x}, \text{MJD})$$

(b) Extract the leap seconds from the data table.

$$\Delta\text{UT1} = y_l$$

end

5. Return the result.

return ΔUT1

Outputs:

- $\Delta\text{UT1} \in \mathbb{R}$ - difference between UT1 (Universal Time 1) and UTC (Universal Coordinated Time) [s]

Note:

- This algorithm, as well as the `find_interval` algorithm from [22], both assume 1-based indexing, and therefore need to be modified for use in programming languages that use 0-based indexing.

Test Cases:

- See Appendix A.2.2.

Leap Seconds

The offset ΔAT represents the accumulated leap seconds used by UTC to stay within $\pm 0.9^{\text{s}}$ of UT1 (this is what causes the difference between TAI and UTC). It is published in IERS Bulletin C. However, leap seconds are rarely added, so it is easiest to manually define the data from Table 5.1 below in your code.

Table 5.1: Leap second history ($\Delta\text{AT} = \text{TAI} - \text{UTC}$) [21], [25], [55, p. 191].

IERS Bulletin C No.	Date	Modified Julian Date	ΔAT [s]
-	1972 January 1	41317	10
-	1972 July 1	41499	11
-	1973 January 1	41683	12
-	1974 January 1	42048	13
-	1975 January 1	42413	14
-	1976 January 1	42778	15
-	1977 January 1	43144	16
-	1978 January 1	43509	17
-	1979 January 1	43874	18
-	1980 January 1	44239	19
-	1981 July 1	44786	20
-	1982 July 1	45151	21
-	1983 July 1	45516	22
-	1985 July 1	46247	23
-	1988 January 1	47161	24
-	1990 January 1	47892	25
-	1991 January 1	48257	26
-	1992 July 1	48804	27
-	1993 July 1	49169	28
10	1994 July 1	49534	29
10–13	1996 January 1	50083	30
13–16, 16	1997 July 1	50630	31
16–30	1999 January 1	51179	32
30–36	2006 January 1	53736	33
36–43	2009 January 1	54832	34
43–49	2012 July 1	56109	35
49–52	2015 July 1	57204	36
52–63	2017 January 1	57754	37

Algorithm 53: `get_dat`

Obtains the difference between TAI and UTC (i.e. leap seconds) ($\Delta\text{AT} = \text{TAI} - \text{UTC}$).

Inputs:

- $\text{MJD} \in \mathbb{R}$ - modified Julian date of any time scale [MJD]

Procedure:

1. Load the ΔAT vs. MJD data from Table 5.1. Store the ΔAT values in the vector $\mathbf{y} = (y_1, ..., y_N)^T$ and the MJD data in the vector $\mathbf{x} = (y_1, ..., y_N)^T$.
2. Find the number of data points, N, given that $\mathbf{x} \in \mathbb{R}^n$.

   $N = \texttt{length}(\mathbf{x})$

3. Edge case: the specified date is after the last available date in the data set.

   if $\text{MJD} \geq x_N$
   $\quad \Delta\text{AT} = y_u$

4. Base case: the specified date is either (a) before the first available date in the data set or (b) contained within the available dates in the data set.

else

(a) Find the lower bound of the interval of $\mathbf{x}$ that contains MJD. This can be done using the `find_interval` algorithm from [22]. Note that this algorithm will automatically return the first interval of $\mathbf{x}$ if $\text{MJD} < x_1$.

$$[l, \sim] = \texttt{find\_interval}(\mathbf{x}, \text{MJD})$$

(b) Extract the leap seconds from the data table.

$$\Delta\text{AT} = y_l$$

end

5. Return the result.

return ΔAT

Outputs:

- $\Delta\text{AT} \in \mathbb{R}$ - leap seconds (difference between TAI (International Atomic Time) and UTC (Universal Coordinated Time) [s]

Note:

- This algorithm, as well as the `find_interval` algorithm from [22], both assume 1-based indexing, and therefore need to be modified for use in programming languages that use 0-based indexing.

Test Cases:

- See Appendix A.2.2.

5.3 Angular Units

Our time systems were originally developed based on the rotational motion of the Earth. Therefore, there are multiple angular quantities (see Section 5.4 that are closely related to the notion of time, and are measured using angular units.

Radians $\leftrightarrow$ Degrees

Let θ^{rad} represent an angle in radians and θ° represent that same angle in degrees. Then

$$\boxed{\theta^{\text{rad}} = \left(\frac{\pi}{180}\right)\theta^{\circ} \qquad \leftrightarrow \qquad \theta^{\circ} = \left(\frac{180}{\pi}\right)\theta^{\text{rad}}} \tag{5.16}$$

Formalizing Eq. (5.16) as Algorithms 54 and 55,

Algorithm 54: `deg2rad`
Degrees to radians.

Inputs:
- $\theta^\circ \in \mathbb{R}$ - angle in degrees $[^\circ]$

Procedure:

$$\theta^{\text{rad}} = \left(\frac{\pi}{180}\right)\theta^\circ$$

return θ^{rad}

Outputs:
- $\theta^{\text{rad}} \in \mathbb{R}$ - angle in radians $[\text{rad}]$

Test Cases:
- See Appendix A.3.

Algorithm 55: `rad2deg`
Radians to degrees.

Inputs:
- $\theta^{\text{rad}} \in \mathbb{R}$ - angle in radians $[\text{rad}]$

Procedure:

$$\theta^\circ = \left(\frac{180}{\pi}\right)\theta^{\text{rad}}$$

return θ°

Outputs:
- $\theta^\circ \in \mathbb{R}$ - angle in degrees $[^\circ]$

Test Cases:
- See Appendix A.3.

Degrees, Arcminutes, and Arcseconds (Angle Measurement)

Arcminutes and arcseconds are subdivisions of a degree, similar to how minutes and seconds are subdivisions of an hour. Arcminutes are denoted by a single apostrophe, while arcseconds are denoted with two apostrophes [55, p. 175].

$$1' = 1 \text{ arcminute} \qquad 1'' = 1 \text{ arcsecond} \tag{5.17}$$

The relationship between the various units is

$$1^\circ = 60' = 3600''$$

Since $1^{\circ} = (\pi/180)$ rad,

$$1^{\circ} = 3600'' = \frac{\pi}{180} \text{ rad} \quad \rightarrow \quad 1 \text{ rad} = \left(\frac{(180)(3600)}{\pi}\right)'' = \left(\frac{648000}{\pi}\right)''$$

Conversely,

$$1'' = \frac{\pi}{648000} \text{ rad}$$

Even smaller units can also be defined by using standard metrix prefixes [27].

$$1 \text{ mas} = 1 \text{ milliarcsecond} = (10^{-3})''$$
$$1 \text{ μas} = 1 \text{ microarcsecond} = (10^{-6})''$$

Typically, we deal with arcseconds. Therefore, we introduce the equations below to convert an angle θ between radians, degrees, and arcseconds.

$$\boxed{\theta'' = (3600)\theta^{\circ} \qquad \leftrightarrow \qquad \theta^{\circ} = \left(\frac{1}{3600}\right)\theta''} \tag{5.18}$$

$$\boxed{\theta'' = \left(\frac{648000}{\pi}\right)\theta^{\text{rad}} \qquad \leftrightarrow \qquad \theta^{\text{rad}} = \left(\frac{\pi}{648000}\right)\theta''} \tag{5.19}$$

Degrees ↔ Arcseconds

Algorithm 56: `deg2arcsec`
Degrees to arcseconds.

Inputs:
- $\theta^{\circ} \in \mathbb{R}$ - angle in degrees [°]

Procedure:

$$\theta'' = (3600)\theta^{\circ}$$

Outputs:
- $\theta'' \in \mathbb{R}$ - angle in arcseconds [″]

Test Cases:
- See Appendix A.3.

Algorithm 57: `arcsec2deg`
Arcseconds to degrees.

Inputs:
- $\theta'' \in \mathbb{R}$ - angle in arcseconds [″]

Procedure:

$$\theta^\circ = \left(\frac{1}{3600}\right)\theta''$$

return θ°

Outputs:

- $\theta^\circ \in \mathbb{R}$ - angle in degrees $[^\circ]$

Test Cases:

- See Appendix A.3.

Radians ↔ Arcseconds

Algorithm 58: `rad2arcsec`
Radians to arcseconds.

Inputs:

- $\theta^{\text{rad}} \in \mathbb{R}$ - angle in radians $[\text{rad}]$

Procedure:

$$\theta'' = \left(\frac{648000}{\pi}\right)\theta^{\text{rad}}$$

return θ''

Outputs:

- $\theta'' \in \mathbb{R}$ - angle in arcseconds $['']$

Test Cases:

- See Appendix A.3.

Algorithm 59: `arcsec2rad`
Arcseconds to radians.

Inputs:

- $\theta'' \in \mathbb{R}$ - angle in arcseconds $['']$

Procedure:

$$\theta^{\text{rad}} = \left(\frac{\pi}{648000}\right)\theta''$$

return θ^{rad}

Outputs:

- $\theta^{\text{rad}} \in \mathbb{R}$ - angle in radians [rad]

Test Cases:
- See Appendix A.3.

Degrees ↔ Degree-Arcminute-Arcsecond

Geographic coordinates are often given in **degree-arcminute-arcsecond (DMS)** format, where the decimal portion of the coordinates (in degrees) are represented using arcminutes and arcseconds. To convert to a pure decimal format, we can use Algorithm 60 below [55, p. 197].

Algorithm 60: `dms2deg`
Degree-minute-second to degrees.

Inputs:
- $\mathrm{d} \in \mathbb{R}$ - degrees [°]
- $\mathrm{m} \in \mathbb{R}$ - arcminutes [′]
- $\mathrm{s} \in \mathbb{R}$ - arcseconds [″]

Procedure:

$$\theta^\circ = \mathrm{d} + \frac{\mathrm{m}}{60} + \frac{\mathrm{s}}{3600}$$

return θ°

Outputs:
- $\theta^\circ \in \mathbb{R}$ - angle in degrees [°]

Test Cases:
- See Appendix A.3.

Algorithm 61: `deg2dms`
Degrees to degree-minute-second.

Inputs:
- $\theta^\circ \in \mathbb{R}$ - angle in degrees [°]

Procedure:
1. Degree portion $[^\circ]^a$.

$$\mathrm{d} = \texttt{fix}(\theta^\circ)$$

2. Auxiliary parameter [°].

$$\alpha = \theta^\circ - \mathrm{d}$$

3. Arcminute portion [′].

$$\mathrm{m} = \texttt{fix}(60\alpha)$$

4. Arcsecond portion [″].

$$\mathrm{s} = 3600\left(\alpha - \frac{\mathrm{m}}{60}\right)$$

5. Return the result.

return $\mathrm{d}, \mathrm{m}, \mathrm{s}$

Outputs:
- $\mathrm{d} \in \mathbb{R}$ - degrees [°]
- $\mathrm{m} \in \mathbb{R}$ - arcminutes [′]
- $\mathrm{s} \in \mathbb{R}$ - arcseconds [″]

Test Cases:
- See Appendix A.3.

[a] The `fix` operation represents rounding towards zero (i.e. truncation).

Radians ↔ Degree-Arcminute-Arcsecond

To convert between radians and degree-arcminute-arcsecond form, we can just use the previous two algorithms and convert either the input or output to degrees.

Algorithm 62: `dms2rad`
Degree-minute-second to radians.

Inputs:
- $\mathrm{d} \in \mathbb{R}$ - degrees [°]
- $\mathrm{m} \in \mathbb{R}$ - arcminutes [′]
- $\mathrm{s} \in \mathbb{R}$ - arcseconds [″]

Procedure:
1. Angle in degrees (Algorithm 60) [°].

$$\theta^\circ = \texttt{dms2deg}(\mathrm{d}, \mathrm{m}, \mathrm{s})$$

2. Angle in radians (Algorithm 54) [rad].

$$\theta^{\mathrm{rad}} = \texttt{deg2rad}(\theta^\circ)$$

3. Return the result.

return θ^{rad}

Outputs:
- $\theta^{\mathrm{rad}} \in \mathbb{R}$ - angle in radians [rad]

Test Cases:
- See Appendix A.3.

Algorithm 63: `rad2dms`
Radians to degree-minute-second.

Inputs:
- $\theta^{\text{rad}} \in \mathbb{R}$ - angle in radians [rad]

Procedure:
1. Angle in degrees (Algorithm 55) [°].

$$\theta^{\circ} = \texttt{rad2deg}(\theta^{\text{rad}})$$

2. Angle in DMS form (Algorithm 61) [°,′,″].

$$[\text{d}, \text{m}, \text{s}] = \texttt{deg2dms}(\theta^{\circ})$$

3. Return the result.

return $\text{d}, \text{m}, \text{s}$

Outputs:
- $\text{d} \in \mathbb{R}$ - degrees [°]
- $\text{m} \in \mathbb{R}$ - arcminutes [′]
- $\text{s} \in \mathbb{R}$ - arcseconds [″]

Test Cases:
- See Appendix A.3.

5.4 Sidereal Time

SI and sidereal seconds

$$t_{\text{UT1}} = t_{\text{UTC}} + \Delta\text{UT1}$$

where ΔUT1 can be found in the IERS Bulletin B.

$$1 \text{ sidereal day} = 86164.0905 \text{ s} = 23 \text{ h } 56 \text{ min } 4.0905 \text{ s} = 23.9344696 \text{ h}$$

$$1 \text{ sidereal s} = 86164.0905 \text{ s} = 23 \text{ h } 56 \text{ min } 4.0905 \text{ s} = 23.9344696 \text{ h}$$

see p. 180 of vallado

5.5 The Time Object

TimeConverter object? Time object?

PART III

Appendices

A

Test Cases

A.1 Rotation Test Cases

A.1.1 Rotation Matrices

Elementary Rotations

θ [rad]	$\mathbf{R}_1(\theta)$ `rot1` *(Algorithm 1)*	$\mathbf{R}_2(\theta)$ `rot2` *(Algorithm 2)*	$\mathbf{R}_3(\theta)$ `rot3` *(Algorithm 3)*
0	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
$\pi/4$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \sqrt{2}/2 & \sqrt{2}/2 \\ 0 & -\sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix}$	$\begin{bmatrix} \sqrt{2}/2 & 0 & -\sqrt{2}/2 \\ 0 & 1 & 0 \\ \sqrt{2}/2 & 0 & \sqrt{2}/2 \end{bmatrix}$	$\begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 & 0 \\ -\sqrt{2}/2 & \sqrt{2}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
$\pi/2$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
$3\pi/4$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -\sqrt{2}/2 & \sqrt{2}/2 \\ 0 & -\sqrt{2}/2 & -\sqrt{2}/2 \end{bmatrix}$	$\begin{bmatrix} -\sqrt{2}/2 & 0 & -\sqrt{2}/2 \\ 0 & 1 & 0 \\ \sqrt{2}/2 & 0 & -\sqrt{2}/2 \end{bmatrix}$	$\begin{bmatrix} -\sqrt{2}/2 & \sqrt{2}/2 & 0 \\ -\sqrt{2}/2 & -\sqrt{2}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
π	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
$5\pi/4$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -\sqrt{2}/2 & -\sqrt{2}/2 \\ 0 & \sqrt{2}/2 & -\sqrt{2}/2 \end{bmatrix}$	$\begin{bmatrix} -\sqrt{2}/2 & 0 & \sqrt{2}/2 \\ 0 & 1 & 0 \\ -\sqrt{2}/2 & 0 & -\sqrt{2}/2 \end{bmatrix}$	$\begin{bmatrix} -\sqrt{2}/2 & -\sqrt{2}/2 & 0 \\ \sqrt{2}/2 & -\sqrt{2}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$3\pi/2$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
$7\pi/4$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \sqrt{2}/2 & -\sqrt{2}/2 \\ 0 & \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix}$	$\begin{bmatrix} \sqrt{2}/2 & 0 & \sqrt{2}/2 \\ 0 & 1 & 0 \\ -\sqrt{2}/2 & 0 & \sqrt{2}/2 \end{bmatrix}$	$\begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 0 \\ \sqrt{2}/2 & \sqrt{2}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
2π	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

3-2-1 Rotation Sequence (`rot321`)

"Forward" Test:

$\theta_1 = 30\left(\frac{\pi}{180}\right)$
$\theta_2 = -40\left(\frac{\pi}{180}\right)$
$\theta_3 = 50\left(\frac{\pi}{180}\right)$
$\texttt{rot321}(\theta_1, \theta_2, \theta_3)$ should equal $\texttt{rot1}(\theta_3)\texttt{rot2}(\theta_2)\texttt{rot3}(\theta_1)$

"Reverse" Test:

$\theta_1 = 30\left(\frac{\pi}{180}\right)$
$\theta_2 = -40\left(\frac{\pi}{180}\right)$
$\theta_3 = 50\left(\frac{\pi}{180}\right)$
$\texttt{rot321}(\theta_1, \theta_2, \theta_3)^T$ should equal $\texttt{rot3}(-\theta_1)\texttt{rot2}(-\theta_2)\texttt{rot3}(-\theta_3)$

3-1-3 Rotation Sequence (`rot313`)

"Forward" Test:

$\theta_1 = 30\left(\frac{\pi}{180}\right)$
$\theta_2 = -40\left(\frac{\pi}{180}\right)$
$\theta_3 = 50\left(\frac{\pi}{180}\right)$
$\texttt{rot313}(\theta_1, \theta_2, \theta_3)$ should equal $\texttt{rot3}(\theta_3)\texttt{rot1}(\theta_2)\texttt{rot3}(\theta_1)$

"Reverse" Test:

$\theta_1 = 30\left(\frac{\pi}{180}\right)$
$\theta_2 = -40\left(\frac{\pi}{180}\right)$
$\theta_3 = 50\left(\frac{\pi}{180}\right)$
$\texttt{rot313}(\theta_1, \theta_2, \theta_3)^T$ should equal $\texttt{rot3}(-\theta_1)\texttt{rot1}(-\theta_2)\texttt{rot3}(-\theta_3)$

`matrotate` (Algorithm 8)

$$\mathbf{R}_{A\to B} = \begin{bmatrix} 0.5721 & 0.4156 & -0.7071 \\ -0.7893 & 0.0446 & -0.6124 \\ -0.2230 & 0.9084 & 0.3536 \end{bmatrix}, \quad [\boldsymbol{r}]_A = \begin{bmatrix} 5 \\ 4 \\ 3 \end{bmatrix}$$

$$[\boldsymbol{r}]_B = \texttt{matrotate}(\mathbf{R}_{A\to B}, [\boldsymbol{r}]_A) = \begin{bmatrix} 2.4016 \\ -5.6053 \\ 3.5794 \end{bmatrix}$$

`matchain` (Algorithm 9)

$$\mathbf{R}_{A\to B} = \begin{bmatrix} 0.5721 & 0.4156 & -0.7071 \\ -0.7893 & 0.0446 & -0.6124 \\ -0.2230 & 0.9084 & 0.3536 \end{bmatrix}, \quad \mathbf{R}_{B\to C} = \begin{bmatrix} -0.5721 & -0.5721 & 0.5878 \\ 0.0064 & 0.7135 & 0.7006 \\ -0.8202 & 0.4046 & -0.4045 \end{bmatrix}$$

$$\mathbf{R}_{A\to C} = \texttt{matchain}(\mathbf{R}_{A\to B}, \mathbf{R}_{B\to C}) = \begin{bmatrix} -0.0068 & 0.2707 & 0.9627 \\ -0.7157 & 0.6709 & -0.1937 \\ -0.6984 & -0.6903 & 0.1892 \end{bmatrix}$$

A.1.2 Conversions Between Rotation Parameterizations

To test the conversions between the different rotation parameterizations, we take a systematic approach; once we verify one algorithm using some unit tests, we can use that algorithm when constructing unit tests to verify other algorithms.

`eul2mat_321` (Algorithm 10)

Note that `eul2mat_321` (Algorithm 10) is just a wrapper around `rot321` (Algorithm 4), where we set $\theta_1 = \psi$, $\theta_2 = \theta$, and $\theta_3 = \phi$. To simply check that we performed the wrapping correctly, we can perform the following test:

$$\psi = 30\left(\frac{\pi}{180}\right)$$
$$\theta = -40\left(\frac{\pi}{180}\right)$$
$$\phi = 50\left(\frac{\pi}{180}\right)$$
$$\texttt{eul2mat\_321}(\psi, \theta, \phi) \quad \Longrightarrow \quad \texttt{rot321}(\psi, \theta, \phi)$$

As an additional check, we can perform the following numerical comparison:

$$\psi = \frac{\pi}{6}$$
$$\theta = -\frac{\pi}{6}$$
$$\phi = \frac{3\pi}{4}$$
$$\texttt{eul2mat\_321}(\psi, \theta, \phi) \quad \Longrightarrow \quad \begin{bmatrix} -0.6124 & 0.6124 & 0.5000 \\ -0.4356 & -0.7891 & 0.4330 \\ 0.6597 & 0.0474 & 0.7500 \end{bmatrix}$$

`mat2eul_321` (Algorithm 11)

After `eul2mat_321` has been unit tested, we can unit test `mat2eul_321` (Algorithm 11) by defining a set of 3-2-1 Euler angles, constructing a rotation matrix from those angles using `eul2mat_321`, and then checking whether `mat2eul_321` correctly recovers those angles.

$$\psi = \frac{3\pi}{4}$$
$$\theta = -\frac{\pi}{6}$$
$$\phi = \frac{\pi}{6}$$
$$\mathbf{R} = \texttt{eul2mat\_321}(\psi, \theta, \phi)$$
$$\texttt{mat2eul\_321}(\mathbf{R}) \quad \Longrightarrow \quad \psi = \frac{\pi}{6}, \quad \theta = -\frac{\pi}{6}, \quad \phi = \frac{3\pi}{4}$$

Numerically,

$$\mathbf{R} = \begin{bmatrix} -0.6124 & 0.6124 & 0.5000 \\ -0.4356 & -0.7891 & 0.4330 \\ 0.6597 & 0.0474 & 0.7500 \end{bmatrix} \quad \Longrightarrow \quad \psi = \frac{\pi}{6}, \quad \theta = -\frac{\pi}{6}, \quad \phi = \frac{3\pi}{4}$$

Next, we can test the pitch-up singularity using a yaw angle of 0, in which case the rotation matrix to 3-2-1 Euler angle conversion should correctly recover the roll angle.

$$\psi = 0$$
$$\theta = \frac{\pi}{2}$$
$$\phi = \frac{\pi}{5}$$
$$\mathbf{R} = \texttt{eul2mat\_321}(\psi, \theta, \phi)$$
$$\texttt{mat2eul\_321}(\mathbf{R}) \quad \Longrightarrow \quad \psi = 0, \quad \theta = \frac{\pi}{2}, \quad \phi = \frac{\pi}{5}$$

Testing the pitch-up singularity with a yaw angle of $-\pi/6$,

$$\psi = -\frac{\pi}{6}$$
$$\theta = \frac{\pi}{2}$$
$$\phi = \frac{\pi}{5}$$
$$\mathbf{R} = \texttt{eul2mat\_321}(\psi, \theta, \phi)$$
$$\texttt{mat2eul\_321}(\mathbf{R}) \quad \Longrightarrow \quad \psi = 0, \quad \theta = \frac{\pi}{2}, \quad \phi = 1.1519$$

Next, we can test the pitch-down singularity using a yaw angle of 0, in which case the rotation matrix to 3-2-1 Euler angle conversion should correctly recover the roll angle.

$$\psi = 0$$
$$\theta = -\frac{\pi}{2}$$
$$\phi = \frac{\pi}{5}$$
$$\mathbf{R} = \texttt{eul2mat\_321}(\psi, \theta, \phi)$$
$$\texttt{mat2eul\_321}(\mathbf{R}) \quad \Longrightarrow \quad \psi = 0, \quad \theta = -\frac{\pi}{2}, \quad \phi = \frac{\pi}{5}$$

Testing the pitch-down singularity with a yaw angle of $-\pi/6$,

$$\psi = -\frac{\pi}{6}$$
$$\theta = -\frac{\pi}{2}$$
$$\phi = \frac{\pi}{5}$$
$$\mathbf{R} = \texttt{eul2mat\_321}(\psi, \theta, \phi)$$
$$\texttt{mat2eul\_321}(\mathbf{R}) \quad \Longrightarrow \quad \psi = 0, \quad \theta = -\frac{\pi}{2}, \quad \phi = 0.1047$$

Next, we need to check the cases where the R_{13} element of the rotation matrix is slightly less than -1 or slightly greater than 1 due to numerical issues near the pitch-up and pitch-down singularities. For the pitch-up case, we construct a rotation matrix with a pitch-up singularity as before, subtract 10^{-14} to the R_{13} element (which should originally be $R_{13} = -1$), and then convert it back to 3-2-1 Euler angles.

$$\psi = -\frac{\pi}{6}$$
$$\theta = \frac{\pi}{2}$$
$$\phi = \frac{\pi}{5}$$
$$\mathbf{R} = \texttt{eul2mat\_321}(\psi, \theta, \phi)$$
$$R_{13} = R_{13} - 10^{-14}$$
$$\texttt{mat2eul\_321}(\mathbf{R}) \quad \Longrightarrow \quad \psi = 0, \quad \theta = \frac{\pi}{2}, \quad \phi = 1.1519$$

Next, we do the same for the pitch-down case, but instead *add* 10^{-14} to the R_{13} element (which should originally be $R_{13} = 1$).

$$\psi = -\frac{\pi}{6}$$
$$\theta = -\frac{\pi}{2}$$
$$\phi = \frac{\pi}{5}$$
$$\mathbf{R} = \texttt{eul2mat\_321}(\psi, \theta, \phi)$$
$$R_{13} = R_{13} + 10^{-14}$$
$$\texttt{mat2eul\_321}(\mathbf{R}) \quad \Longrightarrow \quad \psi = 0, \quad \theta = -\frac{\pi}{2}, \quad \phi = 0.1047$$

Finally, since we have already tested `eul2mat_321`, we can perform a large number of tests for `mat2eul_321` to ensure it returns the original Euler angles.

1. Define many values for $\psi \in [-\pi, \pi]$, $\theta \in (-\pi/2, \pi/2)$, and $\phi \in [-\pi, \pi]$. Make sure not to specify $\theta = \pm\pi/2$; due to the 3-2-1 Euler angle singularity at these points, we will not be able to recover the original quaternion from the 3-2-1 Euler angles.
2. Randomly reorder the values of ψ, θ, and ϕ to ensure we have coverage for all combinations of signs.
3. For each of the values from step 1, calculate the rotation matrix using `eul2mat_321`. Then, recalculate the 3-2-1 Euler angles using `mat2eul_321` and make sure they match the original 3-2-1 Euler angles.

`quat2mat` (Algorithm 23)

The test cases below are adapted from the examples in [34].

$$\mathbf{q} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \implies \mathbf{R} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix} 1 \\ 0.5 \\ 0.3 \\ 0.1 \end{bmatrix} \implies \mathbf{R} = \begin{bmatrix} 0.8519 & 0.3704 & -0.3704 \\ 0.0741 & 0.6148 & 0.7852 \\ 0.5185 & -0.6963 & 0.4963 \end{bmatrix}$$

`eul2quat_321` (Algorithm 26)

A simple numerical test is

$$\psi = \frac{\pi}{6}, \quad \theta = -\frac{\pi}{6}, \quad \phi = \frac{3\pi}{4} \implies \mathbf{q} = \begin{bmatrix} 0.2952 \\ 0.8876 \\ 0.1353 \\ 0.3266 \end{bmatrix}$$

Additionally, since we have already tested `eul2mat_321` and `quat2mat`, we can perform a large number of tests for `eul2quat_321` using the following basic procedure:

1. Define many values for $\psi \in [-\pi, \pi]$, $\theta \in [-\pi/2, \pi/2]$, and $\phi \in [-\pi, \pi]$.
2. Randomly reorder the values of ψ, θ, and ϕ to ensure we have coverage for all combinations of signs.
3. For each of the values from step 1, calculate the rotation matrix using `eul2mat_321`; this represents the "true" rotation matrix.
4. For each of the values from step 1, first calculate the quaternion using `eul2quat_321`, and then calculate the rotation matrix using `quat2mat`; these are the rotation matrices we test against the "true" rotation matrices from step 3.

`mat2quat` (Algorithm 24)

To test `mat2quat`, we can first perform the `quat2mat` tests in reverse, but noting that the outputs of `mat2quat` will be unit quaternions.

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \implies \mathbf{q} = \begin{bmatrix} \sqrt{2}/2 \\ 0 \\ \sqrt{2}/2 \\ 0 \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} 0.8519 & 0.3704 & -0.3704 \\ 0.0741 & 0.6148 & 0.7852 \\ 0.5185 & -0.6963 & 0.4963 \end{bmatrix} \implies \mathbf{q} = \begin{bmatrix} 0.8607 \\ 0.4303 \\ 0.2582 \\ 0.0861 \end{bmatrix}$$

Additionally, since we have already tested `eul2mat_321` and `eul2quat_321`, we can perform a large number of tests for `mat2quat` using the following basic procedure:

1. Define many values for $\psi \in [-\pi, \pi]$, $\theta \in [-\pi/2, \pi/2]$, and $\phi \in [-\pi, \pi]$.
2. Randomly reorder the values of ψ, θ, and ϕ to ensure we have coverage for all combinations of signs.
3. For each of the values from step 1, calculate the quaternion using `eul2quat_321`; this represents the "true" quaternion.
4. For each of the values from step 1, first calculate the rotation matrix using `eul2mat_321`, and then calculate the quaternion using `quat2mat`; these are the quaternions we test against the "true" quaternions from step 3.

`quat2eul_321` (Algorithm 25)

A simple numerical test is

$$\mathbf{q} = \begin{bmatrix} 0.2952 \\ 0.8876 \\ 0.1353 \\ 0.3266 \end{bmatrix} \qquad \Longrightarrow \qquad \psi = \frac{\pi}{6}, \quad \theta = -\frac{\pi}{6}, \quad \phi = \frac{3\pi}{4}$$

Additionally, since we have already tested `eul2quat_321`, we can perform a large number of tests for `quat2eul_321` to ensure it returns the original Euler angles.

1. Define many values for $\psi \in [-\pi, \pi]$, $\theta \in (-\pi/2, \pi/2)$, and $\phi \in [-\pi, \pi]$. Make sure not to specify $\theta = \pm\pi/2$; due to the 3-2-1 Euler angle singularity at these points, we will not be able to recover the original quaternion from the 3-2-1 Euler angles.
2. Randomly reorder the values of ψ, θ, and ϕ to ensure we have coverage for all combinations of signs.
3. For each of the values from step 1, calculate the quaternion using `eul2quat_321`. Then, recalculate the 3-2-1 Euler angles using `quat2eul_321` and make sure they match the original 3-2-1 Euler angles.

`axang2quat` (Algorithm 27)

Next, we unit test `axang2quat` since it is an extremely simple algorithm. We can use the following numerical test

cases:

$$\mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \Phi = 0 \quad \Longrightarrow \quad \mathbf{q} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \Phi = \pi \quad \Longrightarrow \quad \mathbf{q} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{e} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}, \quad \Phi = 0 \quad \Longrightarrow \quad \mathbf{q} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{e} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}, \quad \Phi = \pi \quad \Longrightarrow \quad \mathbf{q} = \begin{bmatrix} 0 \\ -\sqrt{3}/3 \\ -\sqrt{3}/3 \\ -\sqrt{3}/3 \end{bmatrix}$$

$$\mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \Phi = \frac{\pi}{2} \quad \Longrightarrow \quad \mathbf{q} = \begin{bmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{e} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}, \quad \Phi = \frac{\pi}{2} \quad \Longrightarrow \quad \mathbf{q} = \begin{bmatrix} \sqrt{2}/2 \\ -\sqrt{6}/6 \\ -\sqrt{6}/6 \\ -\sqrt{6}/6 \end{bmatrix}$$

$$\mathbf{e} = \begin{bmatrix} 0.1 \\ 0.5 \\ -0.3 \end{bmatrix}, \quad \Phi = \frac{7\pi}{4} \quad \Longrightarrow \quad \mathbf{q} = \begin{bmatrix} 0.9239 \\ -0.0647 \\ -0.3234 \\ 0.1941 \end{bmatrix}$$

`quat2axang` (Algorithm 28)

To test `quat2axang`, we can first perform (most of) the `axang2quat` tests in reverse. Note that the first test below

tests the singularity at $\Phi = 0$.

$$\mathbf{q} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \Longrightarrow \quad \mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \Phi = 0$$

$$\mathbf{q} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad \Longrightarrow \quad \mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \Phi = \pi$$

$$\mathbf{q} = \begin{bmatrix} 0 \\ -\sqrt{3}/3 \\ -\sqrt{3}/3 \\ -\sqrt{3}/3 \end{bmatrix}, \quad \Longrightarrow \quad \mathbf{e} = \begin{bmatrix} -\sqrt{3}/3 \\ -\sqrt{3}/3 \\ -\sqrt{3}/3 \end{bmatrix}, \quad \Phi = \pi$$

$$\mathbf{q} = \begin{bmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \\ 0 \\ 0 \end{bmatrix}, \quad \Longrightarrow \quad \mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \Phi = \frac{\pi}{2}$$

$$\mathbf{q} = \begin{bmatrix} \sqrt{2}/2 \\ -\sqrt{6}/6 \\ -\sqrt{6}/6 \\ -\sqrt{6}/6 \end{bmatrix}, \quad \Longrightarrow \quad \mathbf{e} = \begin{bmatrix} -\sqrt{3}/3 \\ -\sqrt{3}/3 \\ -\sqrt{3}/3 \end{bmatrix}, \quad \Phi = \frac{\pi}{2}$$

$$\mathbf{q} = \begin{bmatrix} 0.3827 \\ 0.1562 \\ 0.7808 \\ -0.4685 \end{bmatrix}, \quad \Longrightarrow \quad \mathbf{e} = \begin{bmatrix} 0.1690 \\ 0.8452 \\ -0.5071 \end{bmatrix}, \quad \Phi = \frac{3\pi}{4}$$

Finally, since we have already tested `axang2quat`, we can perform a large number of tests for `quat2axang`.

1. Define many values for $\Phi \in (0, 2\pi)$, $e_1 \in [-1, 1]$, $e_2 \in [-1, 1]$, and $e_3 \in [-1, 1]$. Make sure not to specify $\Phi = 0$ since this represents an edge case that is tested separately.
2. Normalize every principal rotation vector.
3. Randomly reorder the values of Φ, e_1, e_2, and e_3 to ensure we have coverage for all combinations of signs.
4. For each of the values from step 1, calculate the quaternion using `axang2quat`, and then recover the axis-angle representation using `quat2axang`. Make sure that the recovered axis-angle representations match the original axis-angle representations that we started with.

`axang2mat` (Algorithm 13)

To unit test `axang2mat`, we can use the following numerical test cases:

$$\mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \Phi = 0 \quad \Longrightarrow \quad \mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{e} = \begin{bmatrix} -5 \\ 4 \\ -2 \end{bmatrix}, \quad \Phi = 0 \quad \Longrightarrow \quad \mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{e} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \Phi = \frac{\pi}{2} \quad \Longrightarrow \quad \mathbf{R} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{e} = \begin{bmatrix} 0.1 \\ 0.2 \\ -0.4 \end{bmatrix}, \quad \Phi = \frac{5\pi}{4} \quad \Longrightarrow \quad \mathbf{R} = \begin{bmatrix} -0.6258 & 0.7798 & -0.0166 \\ -0.4546 & -0.3819 & -0.8046 \\ -0.6338 & -0.4960 & 0.5935 \end{bmatrix}$$

Additionally, since we have already tested `axang2quat` and `quat2mat`, we can perform a large number of tests for `axang2mat`.

1. Define many values for $\Phi \in [0, 2\pi]$, $e_1 \in [-1, 1]$, $e_2 \in [-1, 1]$, and $e_3 \in [-1, 1]$.
2. Randomly reorder the values of Φ, e_1, e_2, and e_3 to ensure we have coverage for all combinations of signs.
3. For each of the values from step 1, calculate the quaternion using `axang2quat`. Then, calculate the corresponding rotation matrix using `quat2mat`; this represents the "true" rotation matrix.
4. For each of the values from step 1, calculate the rotation matrix using `axang2mat`; these are the rotation matrices we test against the "true" rotation matrices from step 3.

`mat2axang` (Algorithm 12)

To test `mat2axang`, we can first perform the third and fourth `axang2mat` tests in reverse. Note that we skip the first two tests since the rotation matrix to axis-angle conversion has a singularity at $\Phi = 0$ that we will test separately.

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \Longrightarrow \quad \mathbf{e} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \Phi = \frac{\pi}{2}$$

$$\mathbf{R} = \begin{bmatrix} -0.6258 & -0.4546 & -0.6338 \\ 0.7798 & -0.3819 & -0.4960 \\ -0.0166 & -0.8046 & 0.5935 \end{bmatrix} \quad \Longrightarrow \quad \mathbf{e} = \begin{bmatrix} 0.1 \\ 0.2 \\ -0.4 \end{bmatrix}, \quad \Phi = \frac{3\pi}{4}$$

Next, we test the $\Phi = 0$ singularity. Note that the default behavior for this case is to return $(1, 0, 0)^T$ for the principal rotation vector, since there is no way to determine the principal rotation vector if a rotation does not occur. Additionally, since `axang2mat` has already been tested at this point, we can use it to when defining this test case.

$$\mathbf{e} = (0.2673, 0.5345, 0.8018)^T$$
$$\Phi = 0$$
$$\mathbf{R} = \texttt{axang2mat}(\mathbf{e}, \Phi)$$
$$\texttt{mat2axang}(\mathbf{R}) \quad \Longrightarrow \quad \mathbf{e} = (1, 0, 0)^T, \quad \Phi = 0$$

Next, we test the case where $\Phi = \pi$, which is not a singularity, but does represent an edge case for `mat2axang`.

$$\mathbf{e} = (0.2673, 0.5345, 0.8018)^T$$
$$\Phi = \pi$$
$$\mathbf{R} = \texttt{axang2mat}(\mathbf{e}, \Phi)$$
$$\texttt{mat2axang}(\mathbf{R}) \quad \Longrightarrow \quad \mathbf{e} = (0.2673, 0.5345, 0.8018)^T, \quad \Phi = \pi$$

Finally, since we have already tested `axang2mat`, we can perform a large number of tests for `mat2axang`.

1. Define many values for $\Phi \in (0, 2\pi)$, $e_1 \in [-1, 1]$, $e_2 \in [-1, 1]$, and $e_3 \in [-1, 1]$. Make sure not to specify $\Phi = 0$ or $\Phi = 2\pi$ since these represent edge cases that are test separately.
2. Normalize every principal rotation vector.
3. Randomly reorder the values of Φ, e_1, e_2, and e_3 to ensure we have coverage for all combinations of signs.
4. For each of the values from step 1, calculate the rotation matrix using `axang2mat`, and then recover the axis-angle representation using `mat2axang`. Make sure that the recovered axis-angle representations match the original axis-angle representations that we started with.

`axang2eul_321` (Algorithm 14)

To unit test `axang2mat`, we can use the following numerical test cases:

$$\mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \Phi = 0 \qquad \Longrightarrow \qquad \psi = 0, \quad \theta = 0, \quad \phi = 0$$

$$\mathbf{e} = \begin{bmatrix} -5 \\ 4 \\ -2 \end{bmatrix}, \quad \Phi = 0 \qquad \Longrightarrow \qquad \psi = 0, \quad \theta = 0, \quad \phi = 0$$

$$\mathbf{e} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \Phi = \frac{\pi}{2} \qquad \Longrightarrow \qquad \psi = 0, \quad \theta = \frac{\pi}{2}, \quad \phi = 0$$

$$\mathbf{e} = \begin{bmatrix} 0.1 \\ 0.2 \\ -0.4 \end{bmatrix}, \quad \Phi = \frac{3\pi}{4} \qquad \Longrightarrow \qquad \psi = 2.2471, \quad \theta = 0.0166, \quad \phi = -0.9352$$

Additionally, since we have already tested `axang2mat` and `mat2eul_321`, we can perform a large number of tests for `axang2eul_321`.

1. Define many values for $\Phi \in [0, 2\pi]$, $e_1 \in [-1, 1]$, $e_2 \in [-1, 1]$, and $e_3 \in [-1, 1]$.
2. Randomly reorder the values of Φ, e_1, e_2, and e_3 to ensure we have coverage for all combinations of signs.
3. For each of the values from step 1, calculate the rotation matrix using `axang2mat`. Then, calculate the corresponding 3-2-1 Euler angles using `quat2eul_321`; these represent the "true" 3-2-1 Euler angles.
4. For each of the values from step 1, calculate the 3-2-1 Euler angles using `axang2eul_321`; these are the 3-2-1 Euler angles we test against the "true" 3-2-1 Euler angles from step 3.

`eul2axang_321` (Algorithm 15)

The final rotation parameterization conversion algorithm we need to test is `eul2axang_321`. Recall from Section 4.4.2 that we first use `eul2quat_321` to obtain a quaternion, and then use `quat2axang` to convert that quaternion to the axis-angle representation. Therefore, instead of doing a large suite of a tests, we instead just do a few simple numerical tests to make sure we wrapped around these algorithms correctly.

$$\psi = 0, \quad \theta = 0, \quad \phi = 0 \qquad \Longrightarrow \qquad \mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \Phi = 0$$

$$\psi = \frac{\pi}{4}, \quad \theta = \frac{\pi}{8}, \quad \phi = -\frac{\pi}{6} \qquad \Longrightarrow \qquad \mathbf{e} = \begin{bmatrix} -0.5930 \\ 0.1488 \\ 0.7913 \end{bmatrix}, \quad \Phi = 1.0869$$

A.1.3 Quaternions

`quatmul` (Algorithm 17)

First, let's define the quaternions $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{r}$.

$$\mathbf{p} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \qquad \mathbf{q} = \begin{bmatrix} 1 \\ 0.5 \\ 0.5 \\ 0.75 \end{bmatrix}, \qquad \mathbf{r} = \begin{bmatrix} 2 \\ 1 \\ 0.1 \\ 0.1 \end{bmatrix}$$

We can then define the following simple numerical test cases [37]:

$$\mathbf{p} \otimes \mathbf{p} = \texttt{quatmul}(\mathbf{p}, \mathbf{p}) = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \end{bmatrix}$$

$$\mathbf{p} \otimes \mathbf{q} = \texttt{quatmul}(\mathbf{p}, \mathbf{q}) = \begin{bmatrix} 0.5 \\ 1.25 \\ 1.5 \\ 0.25 \end{bmatrix}$$

$$\mathbf{p} \otimes \mathbf{r} = \texttt{quatmul}(\mathbf{p}, \mathbf{r}) = \begin{bmatrix} 1.9 \\ 1.1 \\ 2.1 \\ -0.9 \end{bmatrix}$$

`quatconj` (Algorithm 16)

$$\mathbf{q} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \implies \mathbf{q}^* = \texttt{quatconj}(\mathbf{q}) = \begin{bmatrix} 1 \\ -2 \\ -3 \\ -4 \end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix} 1 \\ -2 \\ -3 \\ -4 \end{bmatrix} \implies \mathbf{q}^* = \texttt{quatconj}(\mathbf{q}) = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \implies \mathbf{q}^* = \texttt{quatconj}(\mathbf{q}) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

`quatnorm` (Algorithm 18)

$$\mathbf{q} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \implies \|\mathbf{q}\| = \texttt{quatnorm}(\mathbf{q}) = 5.4772$$

$$\mathbf{q} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \implies \|\mathbf{q}\| = \texttt{quatnorm}(\mathbf{q}) = 2$$

$$\mathbf{q} = \begin{bmatrix} 0 \\ 1 \\ -1 \\ -1 \end{bmatrix} \implies \|\mathbf{q}\| = \texttt{quatnorm}(\mathbf{q}) = \sqrt{3}$$

$$\mathbf{q} = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} \implies \|\mathbf{q}\| = \texttt{quatnorm}(\mathbf{q}) = 1$$

`quatnormalize` (Algorithm 20)

First, we can perform a set of simple numerical tests.

$$\mathbf{q} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \implies \frac{\mathbf{q}}{\|\mathbf{q}\|} = \texttt{quatnormalize}(\mathbf{q}) = \begin{bmatrix} 0.1826 \\ 0.3651 \\ 0.5477 \\ 0.7303 \end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \implies \frac{\mathbf{q}}{\|\mathbf{q}\|} = \texttt{quatnormalize}(\mathbf{q}) = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix} 0 \\ 1 \\ -1 \\ -1 \end{bmatrix} \implies \frac{\mathbf{q}}{\|\mathbf{q}\|} = \texttt{quatnormalize}(\mathbf{q}) = \begin{bmatrix} 0 \\ \sqrt{3}/3 \\ -\sqrt{3}/3 \\ -\sqrt{3}/3 \end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} \implies \frac{\mathbf{q}}{\|\mathbf{q}\|} = \texttt{quatnormalize}(\mathbf{q}) = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$

Additionally, since we have already tested `quatnorm` and `eul2quat_321`, we can perform a large number of tests for `quatnormalize` using the following basic procedure:

1. Define many values for $q_0, q_1, q_2, q_3 \in [-10, 10]$.
2. Randomly reorder the values of q_0, q_1, q_2, and q_3 to ensure we have coverage for all kinds of rotations.
3. Normalize each randomly generated quaternion.
4. Take the norm of each of the randomly generated quaternions using `quatnorm`.
5. Check to make sure all normalized quaternions actually have a norm of 1.

`quatinv` (Algorithm 19)

First, we can perform a set of simple numerical tests.

$$\mathbf{q} = \begin{bmatrix}1\\2\\3\\4\end{bmatrix} \implies \mathbf{q}^{-1} = \texttt{quatinv}(\mathbf{q}) = \begin{bmatrix}1/30\\-1/15\\-1/10\\-26/195\end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix}1\\1\\1\\1\end{bmatrix} \implies \mathbf{q}^{-1} = \texttt{quatinv}(\mathbf{q}) = \begin{bmatrix}0.25\\-0.25\\-0.25\\-0.25\end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix}0\\1\\-1\\-1\end{bmatrix} \implies \mathbf{q}^{-1} = \texttt{quatinv}(\mathbf{q}) = \begin{bmatrix}0\\-1/3\\1/3\\1/3\end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix}0\\-1\\0\\0\end{bmatrix} \implies \mathbf{q}^{-1} = \texttt{quatinv}(\mathbf{q}) = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

Additionally, since we have already tested `quatmul`, we can perform a large number of tests for `quatinv` using the following basic procedure:

1. Define many values for $q_0, q_1, q_2, q_3 \in [-10, 10]$.
2. Randomly reorder the values of q_0, q_1, q_2, and q_3 to ensure we have coverage for all kinds of rotations.
3. Find the inverse of each randomly generated quaternion using `quatinv`.
4. Multiply each randomly generated quaternion by its inverse and check to make sure it equals $(1, 0, 0, 0)^T$ (the identity quaternion).

`quatchain` (Algorithm 22)

First, let's just perform a simple numerical test.

$$\mathbf{q}_{A\to B} = \begin{bmatrix}0.1826\\0.3651\\0.5477\\0.7303\end{bmatrix}, \quad \mathbf{q}_{B\to C} = \begin{bmatrix}0.2662\\-0.0690\\-0.3451\\0.8973\end{bmatrix} \implies \mathbf{q}_{A\to C} = \begin{bmatrix}0.3925\\-0.8281\\0.2952\\-0.2701\end{bmatrix}$$

Additionally, since we have already tested `eul2quat_321`, `eul2mat_321`, and `mat2quat`, we can perform a large number of tests for `quatchain` using the following basic procedure:

1. Define many values for $\psi_{A\to B} \in [-\pi, \pi]$, $\psi_{B\to C} \in [-\pi, \pi]$, $\theta_{A\to B} \in [-\pi/2, \pi/2]$, $\theta_{B\to C} \in [-\pi/2, \pi/2]$, $\phi_{A\to B} \in [-\pi, \pi]$, and $\phi_{B\to C} \in [-\pi, \pi]$.
2. Randomly reorder the values of ψ, θ, and ϕ to ensure we have coverage for all combinations of signs.
3. For each pair of rotations:
   (a) $\mathbf{q}_{A\to B} = \texttt{eul2quat\_321}(\psi_{A\to B}, \theta_{A\to B}, \phi_{A\to B})$
   (b) $\mathbf{q}_{B\to C} = \texttt{eul2quat\_321}(\psi_{B\to C}, \theta_{B\to C}, \phi_{B\to C})$
   (c) $\mathbf{q}_{A\to C} = \texttt{quatchain}(\mathbf{q}_{A\to B}, \mathbf{q}_{A\to C})$
   (d) $\mathbf{R}_{A\to B} = \texttt{eul2mat\_321}(\psi_{A\to B}, \theta_{A\to B}, \phi_{A\to B})$
   (e) $\mathbf{R}_{B\to C} = \texttt{eul2mat\_321}(\psi_{B\to C}, \theta_{B\to C}, \phi_{B\to C})$
   (f) $\mathbf{R}_{A\to C} = \mathbf{R}_{B\to C}\mathbf{R}_{A\to B}$
   (g) $[\mathbf{q}_{A\to C}]_{\text{true}} = \texttt{mat2quat}(\mathbf{R}_{A\to C})$
   (h) Check to make sure that $\mathbf{q}_{A\to C}$ is equal to $[\mathbf{q}_{A\to C}]_{\text{true}}$.

`quatrotate` (Algorithm 21)

$$\mathbf{q}_{A\to B} = \begin{bmatrix} 0.7018 \\ -0.5417 \\ 0.1724 \\ 0.4292 \end{bmatrix}, \quad [\boldsymbol{r}]_A = \begin{bmatrix} 5 \\ 4 \\ 3 \end{bmatrix} \quad \Longrightarrow \quad [\boldsymbol{r}]_B = \texttt{quatrotate}(\mathbf{q}_{A\to B}, [\boldsymbol{r}]_A) = \begin{bmatrix} 2.4016 \\ -5.6053 \\ 3.5794 \end{bmatrix}$$

`quatang` (Algorithm 29)

TODO

`quatslerp` (Algorithm 30)

TODO

A.2 Time Test Cases

A.2.1 Time Units

`cal2doy` (Algorithm 31) ↔ `doy2cal` (Algorithm 32)

Date	Day of Year	Description
2022 January 22	22	date in January
2020 March 18	78	date in leap year after January
2020 December 31	366	last day of year in leap year
2022 January 1	1	first day of year
2022 December 31	365	last day of year in non-leap year

These test cases were verified using `https://www.esrl.noaa.gov/gmd/grad/neubrew/Calendar.jsp`.

`cal2mjd` (Algorithm 37) ↔ `mjd2cal` (Algorithm 38)

Gregorian Date	Modified Julian Date
1582 October 15 00:00:00	-100840
1600 January 1 00:00:00	-94553
1600 January 1 06:00:00	-94552.75
1600 January 1 12:00:00	-94552.5
1600 January 1 18:00:00	-94552.25
1858 November 16 18:00:00	-0.25
1858 November 17 00:00:00	0

1858 November 17 06:00:00	0.25
2000 January 1 12:00:00	51544.5
2005 May 24 00:00:00	53514
2006 December 19 00:00:00	54088
2006 December 19 06:00:00	54088.25
2006 December 19 18:00:00	54088.75

These test cases were verified using `https://planetcalc.com/503/` and `https://heasarc.gsfc.nasa.gov/cgi-bin/Tools/xTime/xTime.pl`. Note that some of these test cases also came directly from [28].

Additionally, based on the definitions of `cal2mjd` and `mjd2cal` in Algorithms 37 and 38, respectively, their implementations should be tested to make sure they raise an error if a date before 15 October 1582 (-100840 MJD) is given.

`f2hms` (Algorithm 40)

$(0.524223 \text{ of a day}) = 12{:}34{:}52.867199$ (to 6 decimal places)

`hms2f` (Algorithm 39)

$12 : 34 : 52.890204 = (0.524223 \text{ of a day})$ (to 6 decimal places)

`jd2mjd` (Algorithm 33) ↔ `mjd2jd` (Algorithm 34)

JD	MJD
0	-2400000.5
100	-2399900.5
2400000.5	0
2400100.5	100

`jd2t` (Algorithm 35)

1992 August 20 12:14:00 (JD 2448855.009722222) is equal to -0.073647919 Julian centuries since J2000.0.

This test case was adapted from Example 3-5 in [55, p. 188].

`mjd2f` (Algorithm 41)

Date	Day of Year	Description
1858 November 11 15:50:24	0.66	negative MJD
1858 November 16 15:50:24	0.66	barely negative MJD
1858 November 17 16:04:48	0.67	barely positive MJD

2018 July 22 16:04:48	0.67	positive MJD

`mjd2t` (Algorithm 36)

1992 August 20 12:14:00 (MJD 48854.50972222222) is equal to -0.073647919 Julian centuries since J2000.0.

> This test case was adapted from Example 3-5 in [55, p. 188].

A.2.2 Time Scales

`get_dat` (Algorithm 53)

Date	Modified Julian Date	ΔAT [s]	Description
1972 January 1	41317	10	inside data range
1972 June 30	41498	10	inside data range
1972 July 1	41499	11	inside data range
2005 January 1	53371	32	inside data range
2005 January 2	53372	32	inside data range
2005 December 31	53735	32	inside data range
2006 January 1	53736	33	inside data range
1971 December 31	41316	10	outside data range
1900 January 1	15020	10	outside data range
2017 January 2	57755	37	outside data range
2023 May 7	60071	37	outside data range

`get_dut1` (Algorithm 52)

Date	Modified Julian Date	ΔUT1 [s]	Description
1992 January 1	48622	-0.1251659	inside data range
2004 July 25	53211	-0.4573568	inside data range
2017 December 23	58110	0.2252297	inside data range
1991 December 31	48621	-0.1251659	outside data range

> These test cases were randomly selected from `https://datacenter.iers.org/data/html/finals2000A.data.html`.

A.3 Angle Test Cases

`arcsec2deg` (Algorithm 57) ↔ `deg2arcsec` (Algorithm 56)

θ°	θ''
5.4321°	19555.56
6.515555555555555°	23456

`arcsec2rad` (Algorithm 59) ↔ `rad2arcsec` (Algorithm 58)

θ^{rad}	θ''
0.1 rad	20626.48062470964
0.096962736221907 rad	20000

`deg2dms` (Algorithm 61) ↔ `dms2deg` (Algorithm 60)

$-35^\circ\ -15'\ -53.63'' = -35.264897^\circ$

This test case was adapted from Example 3-8 in [55, p. 198].

`dms2rad` (Algorithm 62) ↔ `rad2dms` (Algorithm 63)

$-35^\circ\ -15'\ -53.63'' = -0.6154886$ rad

This test case was adapted from Example 3-8 in [55, p. 198].

`deg2rad` (Algorithm 54) ↔ `rad2deg` (Algorithm 55)

θ^{rad}	θ°
$-\pi/4$ rad	-45°
0 rad	0°
$\pi/4$ rad	45°
2π rad	360°
4π rad	720°

A.4 Kinematics Test Cases

`vec2skew` (Algorithm 6)

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad \Longrightarrow \quad \mathbf{a}^{\times} = \texttt{vec2skew}(\mathbf{a}) = \begin{bmatrix} 0 & -3 & 2 \\ 3 & 0 & -1 \\ -2 & 1 & 0 \end{bmatrix}$$

`skew2vec` (Algorithm 7)

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad \Longrightarrow \quad \mathbf{a}^{\times} = \begin{bmatrix} 0 & -3 & 2 \\ 3 & 0 & -1 \\ -2 & 1 & 0 \end{bmatrix} \quad \Longrightarrow \quad \mathbf{a} = \texttt{skew2vec}(\mathbf{a}^{\times}) = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Bibliography

[1] *Active and passive transformation*. Wikipedia. Accessed: February 18, 2023. URL: https://en.wikipedia.org/wiki/Active_and_passive_transformation.

[2] *Angle between 2 quaternions*. MathWorks. Accessed: May 7, 2023. URL: https://www.mathworks.com/matlabcentral/answers/415936-angle-between-2-quaternions?s_tid=answers_rc1-2_p2_MLT.

[3] *Angular velocity*. Wikipedia. Accessed: March 25, 2023. URL: https://en.wikipedia.org/wiki/Angular_velocity.

[4] *Attitude Transformations*. VectorNav. Accessed: April 23, 2023. URL: https://www.vectornav.com/resources/inertial-navigation-primer/math-fundamentals/math-attitudetran.

[5] *Axis-angle representation*. Wikipedia. Accessed: April 24, 2023. URL: https://en.wikipedia.org/wiki/Axis-angle_representation.

[6] Martin Baker. *Maths - Conversion Axis-Angle to Euler*. EuclidianSpace. Accessed: April 25, 2023. URL: https://www.euclideanspace.com/maths/geometry/rotations/conversions/angleToEuler/index.htm.

[7] *Basis (linear algebra)*. Wikipedia. Accessed: February 22, 2023. URL: https://en.wikipedia.org/wiki/Basis_(linear_algebra).

[8] Rebecca M. Brannon. *Lecture 12: Three-Dimensional Rotation Matrices*. Physics 216 Lecture Notes (UC Santa Cruz). 2012. URL: http://scipp.ucsc.edu/~haber/ph216/rotation_12.pdf.

[9] Rebecca M. Brannon. *Rotation: A review of theorems involving proper orthogonal matrices referenced to three-dimensional physical space*. Sandia National Laboratories. 2002. URL: https://my.mech.utah.edu/~brannon/public/rotation.pdf.

[10] Matthew Brett. *quat2axang*. Transforms3d. Accessed: May 1, 2023. URL: https://github.com/matthew-brett/transforms3d/blob/main/transforms3d/quaternions.py.

[11] *Coordinate vector*. Wikipedia. Accessed: February 20, 2023. URL: https://en.wikipedia.org/wiki/Coordinate_vector.

[12] *Cross Product*. Wikipedia. Accessed: February 27, 2022. URL: https://en.wikipedia.org/wiki/Cross_product.

[13] Neil Dantam. *Quaternion Computation*. 2014. URL: http://www.neil.dantam.name/note/dantam-quaternion.pdf.

[14] *det(A) = 1 implies A is orthogonal*. Stack Exchange. Accessed: February 19, 2023. URL: https://math.stackexchange.com/questions/2449077/textdeta-1-implies-a-is-orthogonal.

[15] James Diebel. *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*. Stanford University. Accessed: April 29, 2023. 2006. URL: https://www.astro.rug.nl/software/kapteyn-beta/_downloads/attitude.pdf.

[16] *Dynamical time scale*. Wikipedia. Accessed: April 1, 2023. URL: https://en.wikipedia.org/wiki/Dynamical_time_scale.

[17] *Euler Angles*. Academic Flight. Accessed: February 18, 2023. URL: https://academicflight.com/articles/kinematics/rotation-formalisms/euler-angles/.

[18] *Gimbal lock*. Wikipedia. Accessed: April 22, 2023. URL: https://en.wikipedia.org/wiki/Gimbal_lock.

[19] *Gregorian calendar*. Wikipedia. Accessed: January 7, 2022. URL: https://en.wikipedia.org/wiki/Gregorian_calendar.

[20] Jozef C. van der Ha and Malcolm D. Shuster. "A Tutorial on Vectors and Attitude". In: *IEEE Control Systems Magazine* 29.2 (2009), pp. 94–107. DOI: 10.1109/MCS.2008.929426. URL: http://www.malcolmdshuster.com/Pubp_018_073y_J_CSM_Vecs&Att_MDS.pdf.

[21] *IERS Bulletins*. International Earth Rotation and Reference Systems Service. Accessed: January 22, 2022. URL: https://www.iers.org/IERS/EN/Publications/Bulletins/bulletins.html.

[22] Tamas Kis. *Algorithms for Interpolation*. 2022. URL: https://tamaskis.github.io/files/Algorithms_for_Interpolation.pdf.

[23] Andrea L'Afflitto. *L'Afflitto - A Mathematical Perspective on Flight Dynamics and Control*. Springer, 2017.

[24] A. K. Lal. *Ordered Basis*. 2007. URL: https://archive.nptel.ac.in/content/storage2/courses/122104018/node41.html.

[25] *Leap_Second_History.dat*. l'Observatoire de Paris Earth Orientation Center. Accessed: January 22, 2022. URL: https://hpiers.obspm.fr/eoppc/bul/bulc/Leap_Second_History.dat.

[26] Steven J. Leon. *Linear Algebra with Applications*. 9th. London: Pearson, 2015.

[27] *Minute and second of arc*. Wikipedia. Accessed: September 27, 2021. URL: https://en.wikipedia.org/wiki/Minute_and_second_of_arc.

[28] *mjuliandate*. MathWorks. Accessed: April 5, 2023. URL: https://www.mathworks.com/help/aerotbx/ug/mjuliandate.html.

[29] *Moment of inertia*. Wikipedia. Accessed: February 18, 2023. URL: https://en.wikipedia.org/wiki/Moment_of_inertia.

[30] Oliver Montenbruck and Eberhard Gill. *Satellite Orbits – Models, Methods, Applications*. 4th. Berlin, Heidelberg: Springer-Verlag, 2012.

[31] Oliver Montenbruck and Thomas Pfleger. *Astronomy on the Personal Computer*. 4th. Berlin, Heidelberg, New York: Springer-Verlag, 2000.

[32] Gérard Petit and Brian Luzum. "IERS Conventions (2010)". In: *International Earth Rotation and Reference Systems Service (IERS)* IERS Technical Note No. 36 (2010). URL: https://www.iers.org/SharedDocs/Publikationen/EN/IERS/Publications/tn/TechnNote36/tn36.pdf?__blob=publicationFile&v=1.

[33] *Principal Rotation Vector (Axis-Angle Formalism)*. Academic Flight. Accessed: April 23, 2023. URL: https://academicflight.com/articles/kinematics/rotation-formalisms/principal-rotation-vector/.

[34] *quat2dcm*. MathWorks. Accessed: April 29, 2023. URL: https://www.mathworks.com/help/aerotbx/ug/quat2dcm.html.

[35] *Quaternions and spatial rotations*. Wikipedia. Accessed: April 22, 2023. URL: https://en.wikipedia.org/wiki/Quaternion.

[36] *Quaternions and spatial rotations*. Wikipedia. Accessed: April 22, 2023. URL: https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation.

[37] *quatmultiply*. MathWorks. Accessed: May 5, 2023. URL: https://www.mathworks.com/help/aerotbx/ug/quatmultiply.html.

[38] *Rotation matrix*. Wikipedia. Accessed: April 24, 2023. URL: https://en.wikipedia.org/wiki/Rotation_matrix.

[39] *Rotations with Quaternions*. Academic Flight. Accessed: April 23, 2023. URL: https://academicflight.com/articles/kinematics/rotation-formalisms/quaternions/.

[40] Soheil Sarabandi and Federico Thomas. "A Survey on the Computation of Quaternions From Rotation Matrices". In: *Journal of Mechanisms and Robotics* 11.2 (2019). DOI: 10.1115/1.4041889. URL: https://upcommons.upc.edu/bitstream/handle/2117/178326/2083-A-Survey-on-the-Computation-of-Quaternions-from-Rotation-Matrices.pdf?sequence=1.

[41] Soheil Sarabandi and Federico Thomas. "Accurate Computation of Quaternions from Rotation Matrices". In: *Advances in Robot Kinematics*. Springer Proceedings in Advanced Robotics. Springer, 2019, pp. 39–46. DOI: 10.1007/978-3-319-93188-3_5. URL: http://www.iri.upc.edu/files/scidoc/2068-Accurate-Computation-of-Quaternions-from-Rotation-Matrices.pdf.

[42] Ken Shoemake. "Animating Rotations with Quaternion Curves". In: *Journal of Mechanisms and Robotics* 11.2 (2019). DOI: 10.1115/1.4041889. URL: https://upcommons.upc.edu/bitstream/handle/2117/178326/2083-A-Survey-on-the-Computation-of-Quaternions-from-Rotation-Matrices.pdf?sequence=1.

[43] *Show that any orthogonal matrix has a determine 1 or -1*. Stack Exchange. Accessed: February 19, 2023. URL: https://math.stackexchange.com/questions/1172802/show-that-any-orthogonal-matrix-has-determinant-1-or-1.

[44] Malcolm D. Shuster. *A Tutorial on Attitude Kinematics*. 2008. URL: http://www.malcolmdshuster.com/Pubp_022_022y_J_AttKin_MDS.pdf.

[45] *Skew-symmetric matrix*. Wikipedia. Accessed: May 7, 2023. URL: https://en.wikipedia.org/wiki/Skew-symmetric_matrix.

[46] *Slerp*. Wikipedia. Accessed: May 7, 2023. URL: https://en.wikipedia.org/wiki/Slerp.

[47] Harry Smith. *Axes Transformations*. Aircraft Flight Mechanics. Accessed: February 18, 2023. URL: https://aircraftflightmechanics.com/EoMs/EulerTransforms.html.

[48] Brian L. Stevens, Frank L. Lewis, and Eric N. Johnson. *Aircraft Control and Simulation*. 3rd. Hoboken, NJ: John Wiley & Sons, 2016.

[49] Iain W. Stewart. *Advanced Classical Mechanics*. MIT OpenCourseWare, 2016. URL: https://ocw.mit.edu/courses/physics/8-09-classical-mechanics-iii-fall-2014/lecture-notes/MIT8_09F14_full.pdf.

[50] James Stewart. "Chapter 12: Vectors and the Geometry of Space". In: *Calculus*. 8th. Boston, MA: Cengage Learning, 2015, pp. 831–885.

[51] *Tensor*. Wikipedia. Accessed: February 18, 2023. URL: https://en.wikipedia.org/wiki/Tensor.

[52] Ashish Tewari. *Atmospheric and Space Flight Dynamics*. Boston: Birkhäuser, 2007.

[53] *Transport theorem*. Wikipedia. Accessed: March 28, 2023. URL: https://en.wikipedia.org/wiki/Transport_theorem.

[54] *Universal time*. Wikipedia. Accessed: April 1, 2023. URL: https://en.wikipedia.org/wiki/Universal_time.

[55] David A. Vallado. *Fundamentals of Astrodynamics and Applications*. 4th. Hawthorne, CA: Microcosm Press, 2013.

[56] James R. Wertz. *Spacecraft Attitude Determination and Control*. Dordrecht, Holland: D. Reidel Publishing Company, 1978.

[57] *Why is the matrix product of 2 orthogonal matrices also an orthogonal matrix?* Stack Exchange. Accessed: July 5, 2020. URL: https://math.stackexchange.com/questions/1416726/why-is-the-matrix-product-of-2-orthogonal-matrices-also-an-orthogonal-matrix.

[58] Shiyu Zhao. "Time Derivative of Rotation Matrices: A Tutorial". In: *arXiv* (2016). DOI: 10.48550/arXiv.1609.06088.