Linear Algebra

Alexander Koldy

Forest Hills Robotics League

alexanderkoldy.ak@gmail.com

September 18, 2024

Introduction

In this lecture we will be looking at linear algebra from a *FIRST* Tech Challenge (FTC) perspective. Linear algebra is typically taught as a semester-long course at the undergraduate level, so we will not go through many of the intricate mathematical details of the subject. If you are interested in the topic, try 3Blue1Brown's Essence of Linear Algebra Course.

For FTC purposes, we will use linear algebra for organizing our robotics math as well as for setting up and solving systems of linear equations. The topics this lecture will cover are:

- Vectors & matrices
- Common matrix operations
- Linear systems of equations

We will also look at how to apply our math in code via Java's Efficient Java Matrix Library (EJML).

Vectors

Let's start by discussing **vectors**. We can think of a vector as a sort of "position" in n-dimensional space, i.e.,

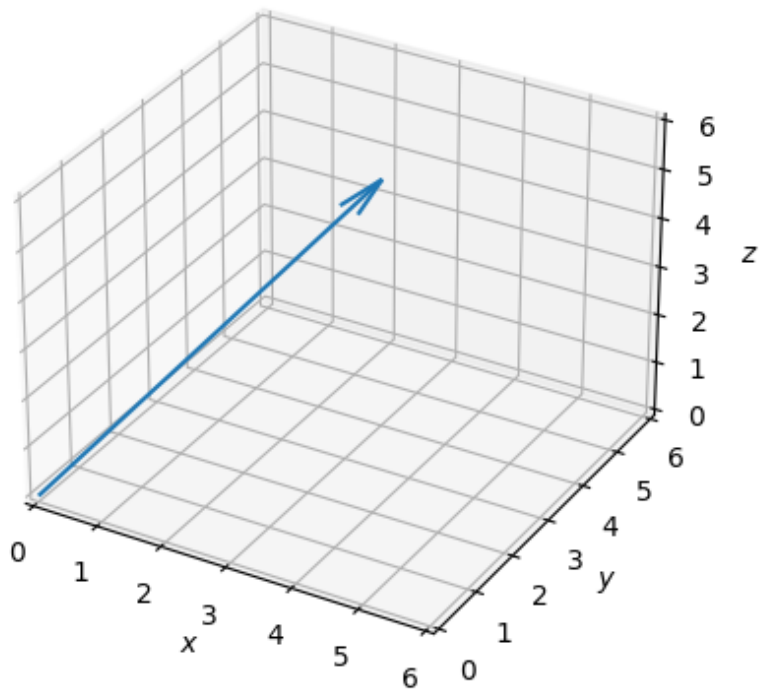
$$\mathbf{v} \in \mathbb{R}^n$$

In English, the expression above reads "$\mathbf{v}$ is a vector in n-dimensional real space".

Let's look an an example where our vector is three-dimensional in Euclidean real space.

$$\mathbf{v} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$$

From the plot on the right, we can see the graphical representation of the vector (denoted by an arrow going from $(0, 0, 0)$ to $(3, 4, 5)$. We can also represent the vector with the point $(3, 4, 5)$, assuming the origin is $(0, 0, 0)$.



Vectors - notation

There are many different ways to write vectors. In terms of symbolic variables, we may often see vectors written as lowercase letters. Sometimes, to distinguish them from single-variable symbols, we write vectors with an arrow above them (i.e., $\vec{v}$) or bolded (i.e., $\mathbf{v}$). For the purposes of our lectures, we will use the bolded notation for the most part.

Additionally, representing the values of the vectors can vary. Let $\mathbf{v}$ be a three-dimensional vector with elements a, b and c. We can represent $\mathbf{v}$ in the following ways:

$$\mathbf{v} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \langle a, b, c \rangle = (a, b, c) = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$$

In our lectures, we will use the matrix notation (first option above) for our vector representations.

Vectors - addition & subtraction

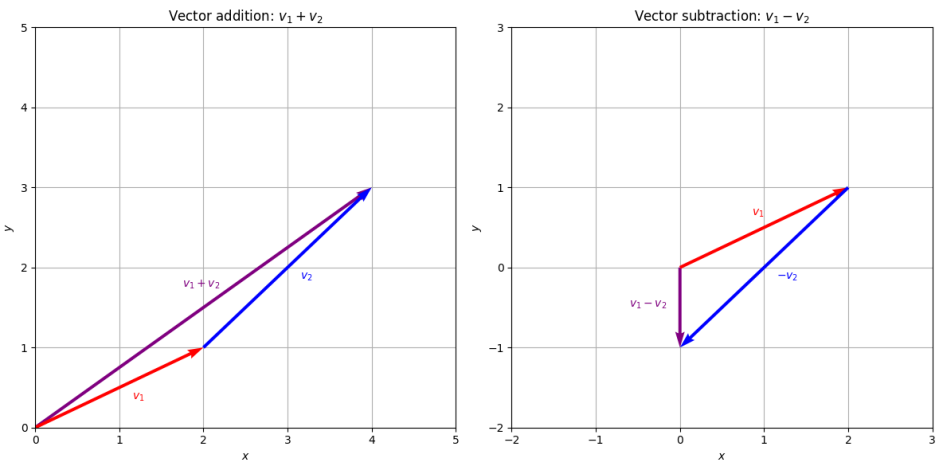
Let's do some basic operations on vectors. Let

$$\mathbf{v}_1 = \begin{bmatrix} a \\ b \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} c \\ d \end{bmatrix}$$

Vector addition is performed by summing the corresponding components of each vector, while vector subtraction is done by subtracting the corresponding components of the second vector from the first vector, i.e.,

$$\mathbf{v_1} + \mathbf{v_2} = \begin{bmatrix} a + c \\ b + d \end{bmatrix}, \quad \mathbf{v_1} - \mathbf{v_2} = \begin{bmatrix} a - c \\ b - d \end{bmatrix}$$

Graphically, we use the vector "tip-to-tail" representation for addition and subtraction. We place the tail (start) of $\mathbf{v}_2$ at the tip (arrow) of $\mathbf{v}_1$, then draw a new vector from the start of $\mathbf{v}_1$ to the tip of $\mathbf{v}_2$. For this graph $a = 1$, $b = 2$. $c = 2$, $d = 2$.



Vectors - norms and dot products

The **norm** of a vector (sometimes referred to as the vector magnitude) is given by

$$|\mathbf{v}| = \sqrt{\sum \mathbf{v}_{(i)}^2}$$

Using the vector $\mathbf{v}_1$ in the slide above, we have:

$$|\mathbf{v}_1| = \sqrt{a^2 + b^2}$$

Notice how similar this looks to the value of the hypotenuse of a triangle.

A vector **dot product** is defined as follows for two vectors of the same size $\mathbf{v}_1$ and $\mathbf{v}_2$:

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = \sum \mathbf{v}_{1,(i)} \mathbf{v}_{2,(i)}$$

This results in a scalar value. As an example, assume the two vectors on the previous slide are our vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. Our dot product is therefore

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = ac + bd$$

Note: when working with vectors, the $\cdot$ symbol represents dot product, not elementwise multiplication. Elementwise multiplication (and division) can be done similarly to addition and subtraction!

Matrices

Let's shift our discussion to **matrices**. In the FTC context, we can use matrices to organize data, solve systems of equations and to apply transformations to vectors. Matrices are typically written as capital letters, but we will write them as bolded capital letters for clarity. Let matrix $\mathbf{A} \in \mathbb{R}^{2\times 3}$ be

$$\mathbf{A} = \begin{matrix} \\ 0 \\ 1 \end{matrix} \begin{matrix} \begin{matrix} 0 & 1 & 2 \end{matrix} \\ \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \end{matrix}$$

This is an example of a matrix of size 2×3, or a matrix with 2 rows and 3 columns. $\mathbf{A}_{(i,j)}$ represents the element at the ith row and jth column. For example

$$\mathbf{A}_{(1,2)} = f$$

Note: we will start our counting at 0 to be consistent with array/matrix indexing in Java

A **square matrix** is a matrix that has the same number of rows and columns!

For the sake of clarity, a vector is a matrix in which one of the dimensions (row or column) is 1 (within a two-dimensional perspective). Therefore we can consider $\mathbf{v}$ to be a matrix!

Matrices

To construct a matrix in Java using EJML, we can use `SimpleMatrix` objects. To construct a known matrix, we can do something like:

```
SimpleMatrix A = new SimpleMatrix(
    new double[][]{
        {1, 2, 3},
        {4, 5, 6},
    }
);
```

This creates a 2×3 matrix. We may also want to create a matrix of zeros. This happens in cases where we know the shape of our matrix, and we plan to fill in the data later. This can be done with:

```
SimpleMatrix A = SimpleMatrix.zeros(2, 3);
```

Let's say we want to set a specific element of a matrix to a new value. If we want to set $\mathbf{A}_{(i,j)} = f$, we can do:

```
A.set(i, j, f);
```

Likewise, obtaining the value at a specific index, i.e., some (i, j) looks like:

```
A.get(i, j);
```

Matrices - transpose

The **transpose** (typically denoted with a $\top$ or T superscript) of a matrix is obtained by swapping its rows with its columns, i.e.,

$$\mathbf{A}^{\top}_{(i,j)} = \mathbf{A}_{(j,i)}, \forall i, j$$

Let's take a look at the matrix $\mathbf{A}^{\top}$:

$$\mathbf{A}^{\top} = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}$$

We see that $\mathbf{A}^{\top}$ is of size 3×2.

Let's also apply this to a vector $\mathbf{v}$. Let

$$\mathbf{v} = \begin{bmatrix} g \\ h \end{bmatrix}$$

We call this a "column vector" since it's row dimension is 1. Let's apply a transpose to $\mathbf{v}$:

$$\mathbf{v}^{\top} = \begin{bmatrix} g & h \end{bmatrix}$$

This is called a "row vector" because it's column dimension is 1.

In EJML, we can simply call the `transpose` function of our `SimpleMatrix` object. Assuming we've constructed $\mathbf{A}$ as a `SimpleMatrix`:

```
SimpleMatrix A_T = A.transpose();
```

Matrices - addition & subtraction

Adding matrices is similar to adding vectors. Let

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

We compute $\mathbf{A} + \mathbf{B}$ using $\mathbf{A}_{(i,j)} + \mathbf{A}_{(i,j)}$ for all i and j. Similarly, we compute $\mathbf{A} - \mathbf{B}$ using $\mathbf{A}_{(i,j)} - \mathbf{A}_{(i,j)}$ for all i and j:

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}, \quad \mathbf{A} - \mathbf{B} = \begin{bmatrix} a-e & b-f \\ c-g & d-h \end{bmatrix}$$

Note: in order to add matrices and vectors, they must have the same size!

To add matrices in EJML we do:

```
SimpleMatrix C = A.plus(B);
```

To subtract, we do:

```
SimpleMatrix C = A.plus(B.negative());
```

Matrices - multiplication

If we are multiplying two matrices $\mathbf{A}$ (of size $a \times b$) and $\mathbf{B}$ (of size $c \times d$), i.e., $\mathbf{C} = \mathbf{AB}$, we have the following:

- The number of columns of $\mathbf{A}$ must equal the number of rows of $\mathbf{B}$, i.e., b must equal c
- The resulting matrix $\mathbf{C}$ will have size $a \times d$ (i.e., number of rows of $\mathbf{A}$ by number of columns of $\mathbf{B}$)
- The (i, j)th element of $\mathbf{C}$ (i.e., $\mathbf{C}_{(i,j)}$) is equal to the dot product between the the ith row of $\mathbf{A}$ and the jth column of $\mathbf{B}$

Let's use the matrices from the previous slide as an example. We see that $\mathbf{A}$ has 2 columns and $\mathbf{B}$ has 2 rows so this is a valid multiplication. Moreover, $\mathbf{A}$ has 2 rows and $\mathbf{B}$ has 2 columns, so $\mathbf{C}$ will be of shape 2×2. The value of $\mathbf{C}$ is therefore

$$\mathbf{C} = \mathbf{AB} = \underset{2 \times 2}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \underset{2 \times 2}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}} = \underset{2 \times 2}{\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}}$$

To take a matrix product in EJML, we can use:

```
SimpleMatrix C = A.mult(B);
```

Matrices - determinants

The **determinant** of a matrix is a single number that tells you if the matrix has an inverse and gives a measure of how the matrix changes the size or volume of geometric shapes.

Note: we only have a valid determinant if the matrix is square

In the case of a 2×2 square matrix, the determinant of some matrix $\mathbf{A} \in \mathbb{R}^{2\times 2}$ is

$$\det(\mathbf{A}) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ab - cd$$

For larger square matrices, the general formula for the determinant some matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is

$$\det(\mathbf{A}) = \sum_{j=1}^{n} (-1)^{i+j} \mathbf{A}_{(i,j)} \det(\mathbf{A}^{(i,j)})$$

where $\mathbf{A}^{(i,j)}$ is the matrix containing the elements of $\mathbf{A}$ with the ith row and jth column removed. In this case i can be any row index of matrix $\mathbf{A}$. For large matrices, this formula may need to be called recursively. We can easily take the determinant in Java using:

```
double det = A.determinant();
```

Matrices - inverses

Let's recall that the inverse of a scalar a can be represented by

$$a^{-1} = \frac{1}{a}$$

Unlike scalars, we cannot divide by a matrix, but the concept of a **matrix inverse** works in a similar fashion. Let's call define the **cofactor matrix** of $\mathbf{A}$ to be $\mathbf{C}$, where

$$\mathbf{C}_{(i,j)} = (-1)^{i+j} \det\left(\mathbf{A}^{(i,j)}\right)$$

Now, the **inverse** of matrix $\mathbf{A}$ is

$$\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})}\mathbf{C}^{\top}$$

In EJML, we can easily take the inverse using:

```
SimpleMatrix A_inv = A.invert();
```

Note: a matrix must be square and it's determinant must be nonzero for a valid inverse to exist. There exists a psuedo-inverse (usually denoted $A^{\dagger}$) for non-square matrices.

Matrices - identity

Just like $a \cdot \frac{1}{a} = \frac{1}{a} \cdot a = 1$, for matrices we have $\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$, where $\mathbf{I}$ is the **identity matrix**. This is a special matrix in which all of the elements are zeros except the diagonals which are ones. A 3×3 identity matrix looks like:

$$\mathbf{I}_{3\times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Any square matrix multiplied by the identity matrix of the same size is itself, i.e.,

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

To create an identity matrix of size $n \times n$ in EJML we use:

```
SimpleMatrix I = SimpleMatrix.identity(n);
```

Systems of linear equations

Let's now apply matrices to something useful! Let's say we have three unknown variables, x, y and z and that we know

$$\begin{aligned} 2x + 5y - z &= 12 \\ y + x &= 5 \\ 2y - 4x + 3z &= -3 \end{aligned}$$

One approach to solve this problem is to isolate a variable and find it's value, then to repeat the process for the remaining variables. However, we can use matrices and vectors to solve this problem in code quite simply!

Note: we have three unknowns and therefore needed three equations to solve this problem

Systems of linear equations

First, let's organize the equations such that the terms with x come first, the terms with y come second and the terms with z come third. Moreover, let's assign a coefficient of 1 and 0 wherever they are missing:

$$2x + 5y + -1z = 12$$
$$1x + 1y + 0z = 5$$
$$-4x + 2y + 3z = -3$$

Now, we will place all of the coefficients into a matrix $\mathbf{A}$, all of the unknowns into vector $\mathbf{x}$ and the constants on the right hand side of the equations into vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} 2 & 5 & -1 \\ 1 & 1 & 0 \\ -4 & 2 & 3 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x \\ y \\ z \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} 12 \\ 5 \\ -3 \end{bmatrix}}_{\mathbf{b}}$$

Now, as long as $\mathbf{A}$ is invertible, we can solve $\mathbf{x}$:

$$\mathbf{A}^{-1}\mathbf{A}\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$
$$\mathbf{I}\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$
$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

Systems of linear equations

Let's set this problem up in EJML and see what we get.

```
SimpleMatrix A = new SimpleMatrix(
    new double[][]{
        {2, 5, -1},
        {1, 1, 0},
        {-4, 2, 3},
    }
);
SimpleMatrix A = new SimpleMatrix(
    new double[][]{
        {12}
        {5}
        {-3}
    }
);
SimpleMatrix x = A.invert().mult(b)
```

Systems of linear equations

After solving for $\mathbf{x}$ we are left with:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 3.4\overline{6} \\ 1.5\overline{3} \\ 2.6 \end{bmatrix}$$

We have successfully solved our problem in only a few lines of code!