01_vectors

August 31, 2023

1 PHZ1140C Vectors

1.1 Tips

- Use **esc r** to disable a cell
- Use **esc y** to reactivate it
- Use **esc m** to go to markdown mode
- Shift + return to execute a cell

Click here for an excellent Python online tutorial.

1.2 Goal

The purpose of this notebook is to help you become familiar with vectors.

1.3 What is a vector?

Many quantities in physics including **displacement**, **velocity**, and **force** can be represented as vectors.

Consider, for example, displacement. As the name implies, displacement has something to do with starting at one place and ending up somewhere else. Suppose you started in Tallahassee, Florida, and ended up in Atlanda, Georgia. Perhaps you flew from Tallahassee to Atlanta, or perhaps you decided to drive. Perhaps you first drove east on the I10 and then north on the I75; or perhaps you chose to drive on state roads and then join the I75 further north.

There are many paths you could have taken to go from Tallahassee to Atlanta. But of all the paths between your starting and end points there is one that is the shortest. This is the straight line from your starting point to your end point. In practice, we can't take this path because a straight tunnel does not exist between Tallahassee and Atlanta!

By definition, displacement is a mathematical object that represents the operation of starting at one point and ending at another by following the straight line between the two points. A displacement, therefore, has two attributes: a **magnitude**, in this case the straight-line distance between the two points, and a **direction**. A **vector** is a quantity that encodes a magnitude, which is always positive or zero, and a direction.

1.4 Representing and encoding a vector

Suppose that v represents the magnitude of a vector. To remind us that a vector also has a direction, we use $\vec{v}$ as the symbolic representation of the vector whose magnitude is v. Another conventon is

to use bold type, v, to represent a vector. To remind us that a magnitude cannot be negative, we sometimes represent the magnitude of a vector using the symbol $|\vec{v}|$ or $|v|$.

An arrow is a conceptually, and visually, appealing way of encoding the two attributes of a vector. The length of the arrow, which obviously can't be negative, is the magnitude, while the direction of the arrow is the direction of the vector.

Another obvious way to encode, or represent, a vector is as a **tuple**, (m, d), where m is the magnitude and d is the direction. (In mathematics, an n-tuple, $(x_1, \cdots, x_n)$, is an ordered sequence of entities x_1 to x_n that are to be viewed as a single entity.) For example, a velocity $\vec{v}$ could be encoded as follows: $\vec{v} = (30 \text{ mph}, 15^{\text{o}} \text{ East of North})$.

Vectors can be multiplied or divided by a number, that is, scaled. Suppose we multiply a vector by a positive number f. We need to agree on what we want this to mean. Let's agree that this means we scale the *magnitude* of the vector by the number f, while leaving the direction unchanged. This means that the new and old vectors would be parallel to each other since they are pointing in the same direction. Going back to our velocity example, this would mean that the new vector $\vec{w}$ is given by

$$\begin{aligned} \vec{w} &= f\,\vec{v}, \\ &= (f \times 30 \text{ mph}, 15^{\text{o}} \text{ East of North}). \end{aligned} \tag{1}$$

Unfortunately, the equation above illustrates a problem with our tuple representation. Since we agree only to change the magnitude, we are forced to treat the first entry of the tuple differently from the second entry. There is another problem.

Notice that if we scale a vector by the inverse of its magnitude, that is, we compute $\vec{v}/v$, then the resulting vector will have a magnitude of one and have the same direction as the vector $\vec{v}$. Moreover, the magnitude of the scaled vector will be dimensionless, that is, it will have no units. A dimensionless vector with a magnitude of one is called a **unit** vector. It is represented with a symbol decorated with a "hat", e.g., $\hat{v}$.

The third problem with our tuple representation of a vector is that the magnitude may or may not be dimensionless, while the direction is always dimensionless. Yet another problem with the tuple representation is that if we scale the vector by a negative number, f, our previous rule for creating a new vector no longer works because magnitudes cannot be negative! So we need to change our rule and scale the magnitude by the *absolute* value of the scale factor f, denoted by the symbol $|f|$.

But what do we do with the negative sign? Well, let's suppose it makes sense to add and subtract vectors. Then, if $f = -1$, it is surely natural to require that $\vec{A} + f\,\vec{A} = 0$. That $\vec{A} - \vec{A} = 0$ surely makes a lot of sense. But what precisely does it mean?

Suppose we go from point A to point B in a straight line, a displacement we'll denote by $\vec{D}$. If now we go from B to A that is a displacement in exactly the opposite direction to $\vec{D}$. Therefore, it makes sense to interpret the vector $-\vec{D}$ as a vector of the same magnitude as $\vec{D}$ but pointing in exactly the opposite direction. Clearly, if I go from A to B and then B to A, my net displacement is zero because it is as if I haven't moved! Therefore, it indeed makes sense that $\vec{D} - \vec{D} = 0$. In summary, if we scale a vector by the number f our rule must be this: scale the magnitude by $|f|$, that is, a positive number, thereby keeping the magnitude positive, but leave the direction unchanged unless f is negative in which case reverse the direction of the vector.

This rule is a bit messy! It would be nice to find a representation of a vector, inspired by its arrow representation, which treats the magnitude and direction in a tidy, uniform, way.

1.5 Vector algebra

There is such a representation, namely, an n-tuple, where for the vectors we'll be using $n = 3$. (Physicists, routinely deal with vectors in $n > 3$ dimensions, even $n = \infty$!) The key point about vectors, as alluded to, is that they can be manipulated algebraically, that is, there exists an algebra of vectors. Vectors can be scaled, they can be added, subtracted, and multiplied. For example, if I go from point A to point B and then from point B to point C, it is the same as going from point A to C. Therefore, if $\vec{a}$ is the displacement from A to B, $\vec{b}$ the displacement from B to C and $\vec{c}$ the displacement from A to C, we define vector addition by

$$\vec{c} = \vec{a} + \vec{b}. \tag{2}$$

In terms of the arrow representation of vectors, this means we start with an arrow whose tail is at A and its head at B, another with its tail at B and its head at C, while the arrow that represents $\vec{c}$ has its tail at A and its head at C, as depicted in the figure to the left. But now we come to a crucial observation about vector addition. Since a vector has two attributes only, magnitude and direction, we are free to move the vector around so long as it maintains its direction and its magnitude (think of moving the arrows around while maintaining their directions). Therefore, we can go from A to C with a displacement along the vector $\vec{b}$ first followed by a displacement along the vector $\vec{a}$, as shown in the figure to the right.

Obviously, we won't go via point B, but we'll arrive at the same place, namely, point C. This sequence of vector operations corresponds to the vector addition

$$\vec{c} = \vec{b} + \vec{a}. \tag{3}$$

This shows that the addition of vectors behaves like the addition of numbers in that the operation of addition is commutative, e.g.: $2 + 3 = 3 + 2$.

1.5.1 Coordinate vectors

The fact that vectors can be added and subtracted suggests a better algebraic way to represent them. Suppose we have designated some point in space to be a marker with respect to which the positions of all other points are specified. The special marker is called the **origin**. Since the choice of origin is arbitrary, we are free to choose it as we see fit. For example, if we are modeling the orbits of the inner planets, we may choose the center of the Sun as the origin. If we are modeling a projectile, it may be more convenient to choose the launch position as the origin. At the origin, we erect three mutually orthogonal lines, that is, lines such that each is at right angles to the other two. Let's label these lines, each an axis, x, y, and z. This construct is called a **Cartesian coordinate system**. We need three axes because space has 3 dimensions. A space with n dimensions needs n axes.

Another way of thinking about the unit vectors is that they *define* the Cartesian coordinate system by specifying the directions of its three axes, as depicted in the figure. The location of any point in 3-dimensional space can be specified by a 3-tuple (x, y, z), where x, y, and z are called the **coordinates** of the point with respect to the origin of the chosen coordinate system along the directions specified by the associated unit vectors. The 3-tuple specifies, or defines, a **position vector**, $\vec{r} = (x, y, z)$. An example of a widely used Cartesian coordinate system is the **heliocentric ecliptic coordinate system**, which is shown here. The center of this coordinate system is near the center of the Sun. In fact, the origin of this coordinate system is (by international agreement) at the center of mass of the Solar System.

Like position in 3-dimensional space, a velocity can also be specified by giving three numbers $\vec{v} = (v_x, v_y, v_z)$, where v_x, v_y, and v_z are the components of the velocity in the $\hat{i}$, $\hat{j}$, and $\hat{k}$ directions, respectively. If we represent the three unit vectors as $\hat{i} = (1, 0, 0)$, $\hat{j} = (0, 1, 0)$, and $\hat{k} = (0, 0, 1)$, we can write position and velocity as

$$\vec{r} = x\,\hat{i} + y\,\hat{j} + z\,\hat{k}, \tag{4}$$

$$\vec{v} = v_x\,\hat{i} + v_y\,\hat{j} + v_z\,\hat{k}. \tag{5}$$

But remember the above expressions remain true regardless of the specific coordinate representations of the vectors.

As noted above, vectors (if you think of them as arrows) can be moved around. This is true also of the position vector. But, by convention, we agree to fix the tail end of the position vector at the origin.

An important point to note is that if we change the coordinate system (perhaps it it rotated), the three numbers that represent a vector, that is, the **vector components**, will, in general, change. But the vector they represent remains the same: it has same magnitude and the same direction. A vector is said to be a **coordinate-independent** object. Even though physicists almost always use coordinate systems for calculations, we prefer to use them with coordinate-independent objects such as vectors so that we obtain results that do not depend on the arbitrarily chosen coordinate system. After all, Nature doesn't care about artifacts such as coordinate systems! Vector components are used only because they are a convenient way to define, or represent, a vector, but in many physics problems it is possible to work with vectors without ever having to consider their components. In one of the problems below, you'll be invited to work out a coordinate-independent formula for a new vector from two other vectors.

1.5.2 Import Modules

Make Python modules (that is, collections of programs) available to this notebook.

[2]:
```
import os, sys
import numpy as np
import matplotlib as mp
import matplotlib.pyplot as plt
import sympy as sm
import scipy as sp
#import pandas as pd
#import vpython as vp
#import itertools as it

sm.init_printing()        # activate "pretty printing" of symbolic expressions
%matplotlib inline
```

1.5.3 Setup fonts

[2]:
```
# update fonts
FONTSIZE = 14
font = {'family' : 'sans-serif',
        'weight' : 'normal',
```

```
        'size'   : FONTSIZE}
mp.rc('font', **font)

# use latex if available on system, otherwise set usetex=False
mp.rc('text', usetex=True)

# set a seed to ensure reproducibility
# on a given machine
seed = 314159
rnd  = np.random.RandomState(seed)
```

1.6 Vector Basics

1.6.1 Dot product

Given vectors $\vec{A} = (A_x, A_y, A_z)$ and $\vec{B} = (B_x, B_y, B_z)$ with components defined with respect to some coordinate system, the **dot product** between them is

$$\vec{A} \cdot \vec{B} = A_x B_x + A_y B_y + A_z B_z. \tag{6}$$

At face value, this formula suggests that the dot product depends on the vector components and consequently on the coordinate system that is used to define them. In fact, the dot product is independent of the coordinate system. Try to show this by proving that the dot product can be expressed as

$$\vec{A} \cdot \vec{B} = AB\cos(\theta), \tag{7}$$

where A and B are the magnitudes (see below) and θ is the angle between the vectors. Since neither the magnitudes nor the angle between the vectors depend on the coordinate system, it follows that the dot product is likewise coordinate independent. Therefore, if we use the first form of the dot product, we'll always get the same answer whatever coordinate system we use.

1.6.2 Magnitude

The magnitude of a vector, $\vec{A}$, is defined by

$$A = \sqrt{\vec{A} \cdot \vec{A}}, \tag{8}$$

$$= \sqrt{A_x^2 + A_y^2 + A_z^2}. \tag{9}$$

What ancient theorem does this remind you of?

1.6.3 Cross product

Given two vectors $\vec{A}$ and $\vec{B}$ the cross product creates another vector $\vec{C}$ defined by

$$\vec{C} = \vec{A} \times \vec{B}, \tag{10}$$

$$= AB\sin(\theta)\,\hat{n}, \tag{11}$$

where $\hat{n}$ is a unit vector that is perpendicular to the plane containing $\vec{A}$ and $\vec{B}$. But this leaves a twofold ambiguity in the orientation of $\hat{n}$, which can be resolved as follows. With your right hand do the following.

1. Point your fingers in the direction of vector $\vec{A}$, the vector to the left in the cross product.
2. Curl your fingers into the direction of vector $\vec{B}$. If you can't, you must rotate your wrist by about 180° (yes, this can be quite awkward) and start again with step 1.
3. The direction in which your thumb is pointing gives the correct orientation of $\hat{n}$.

The rule above is called the **right-hand rule**. Remember it; you'll use it many times over the next few years. Like the dot product, the cross product is coordinate-independent. But unlike the dot product, the cross product is non-commutative: $\vec{A} \times \vec{B} \neq \vec{B} \times \vec{A}$.

1.7 Numpy Basics

An **array** is an ordered sequence of objects. For example, [1,2,3,4] is a 1-dimensional array of integers; [[1.5,2.7,3.141,4.2], [0.5,0.1,9.1,3.6]] is an example of a 2-dimensional array of real numbers. **numpy** is the standard software module for manipulating arrays of numbers in Python. The best way to learn numpy and any software is to use it! Work through the following simple examples and try to understand what is being done. Feel free to change the examples to test your understanding.

Then work on **Exercise 1**.

[3]:
```
# 1. creating arrays
a = np.array((1,2,3))
b = np.array((3,2,1))
a, b
```

[3]:
```
(array([1, 2, 3]), array([3, 2, 1]))
```

[4]:
```
# 2. compute a + b
c = a + b
c
```

[4]:
```
array([4, 4, 4])
```

[5]:
```
# 3. compute a*b
d = a * b
d
```

[5]:
```
array([3, 4, 3])
```

[6]:
```
# 4. compute sum(a*b)
f = np.sum(d)
f
```

[6]:
```
10
```

[7]:
```
# 5. divide a by 2
a2 = a / 2
a2
```

[7]:
```
array([0.5, 1. , 1.5])
```

[8]:
```
# 6. compute sqrt(a)
sqa = np.sqrt(a)
sqa
```

[8]:
```
array([1.        , 1.41421356, 1.73205081])
```

[9]:
```
# 7. compute outer product of a and b
ab = np.outer(a, b)
ab
```

[9]:
```
array([[3, 2, 1],
       [6, 4, 2],
       [9, 6, 3]])
```

[10]:
```
# 8. compute inner product of a and b
abi = np.inner(a, b)
abi
```

[10]:
```
10
```

[11]:
```
# 9. convert b to a 2d array of shape (3, 1)
b2 = b[:, np.newaxis]
b2, b2.shape
```

[11]:
```
(array([[3],
        [2],
        [1]]),
 (3, 1))
```

10. compute $a - b2$ and try to explain the result.

[12]:
```
# 10. compute a - b2
ab2 = a - b2
ab2
```

[12]:
```
array([[-2, -1,  0],
       [-1,  0,  1],
       [ 0,  1,  2]])
```

[13]:
```
# 11. compute cos(a)
cosa = np.cos(a)
cosa
```

[13]:
```
array([ 0.54030231, -0.41614684, -0.9899925 ])
```

[14]:
```
# 12. compute acos(a) and convert the angles from radians to degrees
acosa = np.arccos(cosa) * 180 / np.pi
acosa
```

[14]:
```
array([ 57.29577951, 114.59155903, 171.88733854])
```

[15]:
```
# 13. compute dot product a . b
adotb = np.dot(a, b)
adotb
```

[15]:
```
10
```

[16]:
```
# 14. compute cross product a x b
acrossb = np.cross(a, b)
acrossb
```

[16]:
```
array([-4,  8, -4])
```

[17]:
```
# 15. compute cross product of b x a
bcrossa = np.cross(b, a)
bcrossa
```

[17]:
```
array([ 4, -8,  4])
```

[18]:
```
# 16. compute dot product of a and a x b
adot_axb = np.dot(a, acrossb)
adot_axb
```

[18]:
```
0
```

1.8 Exercise 1

Create two vectors $\vec{A} = (2, 3, 6)$ and $\vec{B} = (6, 4, 12)$ using the **numpy** Python module and then answer the following questions. Once the vectors are created, do not work with their components but do everything using **numpy** operations.

1. Compute the magnitude of each vector.
2. Compute the unit vectors associated with each vector.
3. Compute the dot product between the vectors using their components.
4. Compute the angle between the vectors in radians.
5. Compute $\vec{C} = A + B$.
6. Compute the dot product of $\vec{C}$ with itself.
7. Compute $A^2 + B^2 + 2\vec{A} \cdot \vec{B}$. How does this compare with C^2?
8. Compute a vector $\vec{D}$ that is perpendicular to $\vec{A}$. First derive a coordinate-independent formula for $\vec{D}$. (*Hint*: draw a picture with the three vectors represented as arrows.) The vector $\vec{D}$ should start from the vector $\vec{A}$ and touch the head of vector $\vec{B}$. What value should you get if you compute the dot product of $\vec{D}$ with $\vec{A}$? Verify your answer to four significant figures.
9. Compute the vector $\vec{D}$, but this time use cross products (and, at most, a single dot product). (*Hint*: use the right hand rule, starting with $\vec{B} \times \vec{A}$.)

[ ]:
```

```

[ ]:
```

```

[ ]: