# VPython Class 2: Functions, Fields, and Forces

## 1. Getting started

Our goal today will be to create force functions that will accelerate charged particles (via Newton's 2nd Law) in any combination of electric and magnetic fields. I would like you to force function(s) to do two things:

1. simulate a physical scenario for which you already know the resulting motion, based on material in Chapter 26 of Wolfson, and
2. simulate a physical scenario for which you do not already know the resulting motion.

You can start from scratch, or you can modify a copy of your program from last week. As always, the first cell of your program should contain

```
from vpython import *
from math import *
```

## 2. Defining functions

A python function is a lot like a math function — it takes one or more inputs called arguments, does something with them, and typically "returns" an output.

In your Jupyter notebook define a function as follows:

```
def my_function(r):
    # lots of calculations could go here
    return 6*r

print( my_function(2) )
```

Look at the output. Now try

```
x = 7
r = 4
print( my_function(x) )
```

Is the result what you expected? Explain in your own words what the role of the variable `r` is in the function.

There's more than meets the eye with this function. We didn't tell the function what type of variable `r` was. Let's make a vector:

```
v = vector(1,2,0)
print( my_function(v) )
```

Now look at the output. The function `my_function` will return a number when the argument `r` is a number, but it returns a vector when the argument is a vector. You can cut the cells with these practice functions from your notebook.

## 3. Electromagnetic force

If the electric field $\vec{E}(\vec{r})$ and the magnetic field $\vec{B}(\vec{r}$ are known, the force on a particle with a charge $q$ at position $\vec{r}$ and traveling with velocity $\vec{v}$ is

$$\vec{F} = q\left[\vec{E}(\vec{r}) + \vec{v} \times \vec{B}(\vec{r})\right]. \tag{1}$$

### 3.1 Creating a particle

Let's create a charged particle `p1` as a sphere object:

```
p1 = sphere(pos = vector(0, 5, 0), radius = 0.5, color = color.red,
                 vel = vector(5, 0, 0),  mass = 1,  charge = 1)
```

We've named this particle `p1` in case we want to make a second one later. The VPython sphere object has some built-in attributes such as position, radius, and color, and we added some of our own: velocity, mass, and charge. Thus it matches our notion of a charged particle. Let's also add a wall to give some visual perspective:

```
wall = box(pos = vector(0,0,0), size = vector(0.2, 10, 10),
           color = color.blue)
```

### 3.2 Writing a force function

Define a python function that returns the electromagnetic force on a particle. It is conventional to put all function definitions at the beginning of a program, immediately after the `import` statements.

```
def force(particle):
    return particle.charge * ( e_field + cross(particle.vel, b_field) )
```

Here the function `cross()` takes the cross product of two vectors. Compare the python expression in the function to Eq. (1) above. (Think briefly about how you'd write this if you had to specify all the components. Now be grateful VPython saves you from having to do this!)

What is the argument for this force function? It can be any object, e.g., a sphere or a box or a cylinder. The only restriction is that it must be an object with defined charge and velocity vector (so that `particle.charge` and `particle.vel` have values).

Of course, this function won't work unless the vectors `e_field` and `b_field` are assigned values. Let's consider a uniform magnetic field in the $\hat{k}$ direction:

```
e_field = vector(0, 0, 0)
b_field = vector(0, 0, 1)
```

The magnitude of $\vec{B}$ is set to one for simplicity. **Test your force function:**

```
print( force(p1) )
```

Do you get the right answer?

## 4. Animating the particle's motion

Charges in a uniform magnetic field with $\vec{B}$ perpendicular to the charged particle's velocity go in circles. We will let the charge move in a half circle and slam into the wall. Define the time step `dt = 0.01`. Now we use a `while` loop to keep updating the particle's position and velocity until it crashes:

```
# Here is Newton's 2nd Law:
accel = _________________________

while p1.pos.x > -0.01:
    rate(100)

    # Now we update the position and velocity
    p1.pos = _________________________
    p1.vel = _________________________
    # Here is Newton's 2nd Law (again):
    accel = _________________________
```

You fill in the blanks! Why did we make the condition $x > -0.01$? Run the simulation and see if it does what it is supposed to. Pay attention to where we are using vectors and where we are using a vector component.

## 5. Adding a Particle Track

Sometimes it is helpful to see the track of the particle, like the bubble chamber particle reactions you analyzed in PHYS 211 lab. This is fairly easy to add using the `curve` object, which takes a list of position vectors and connects them with lines. If that list of position vectors is all the previous coordinates of the charge, then `curve` will draw the particle track. Before the `while` loop, when you are defining the charge object, add a line

```
track1 = curve()
```

Now we just need to update the list as the particle moves. We can use the `append` method to do this. Inside the `while` loop, include a line

```
    track1.append(pos = p1.pos)
```

This adds the current position of `p1` to the list stored in `track1`. Run this and see what you get.

## 6. Assignment

1. Compare the results of your animation with the analytical results from Section 26.3 in Wolfson. This comparison should be quantitative.

2. Change an attribute (or attributes) of your particle, and re-run your animation. Are your results consistent with the analytical results in Section 26.3 in Wolfson.

3. Change the magnetic field to

   (a) $\vec{B} = \hat{\jmath} + \hat{k}$, and then

   (b) $\vec{B} = \hat{\imath} + \hat{k}$.

   Can you understand the results of your animation?

4. If you have time, set your electric field to $\vec{E} = \hat{\jmath}$, and your magnetic field to $\vec{B} = \hat{k}$, and run your simulation.

## 7. Save and submit

Save the program with the naming convention

`yourlastname_labXX.ipynb`,

where XX is the two-digit assignment number (in this case 02) and "_" is the underscore character. If the program you wish to submit doesn't have this name, please rename your program before submission.

Once you've saved a copy of your code with this filename you can submit it by copying it into my drop box (`facultystaff/m/mligare`). (You can't save it directly there, but you can copy or drag the file into it.)