Physics Modeling in Python

William D. Piñeros
Department of Physics and Materials Science, University of Luxembourg

1 Introduction

In physics, like any science, one observes phenomena and, ideally, carries out experiments to test hypothesis explaining it. But what if an experiment cannot be carried out because, for instance, it is too expensive or technologically difficult? Instead, we can consider *mathematical models*, rooted in physics, which we can test to see if it captures the essential elements of an observed phenomenon. In such a case we could obtain exact solutions to a problem (which we call "analytical") and make limiting predictions, like what happens for a very short times, or when a variable dominates over the others etc.

But even here sometimes the mathematics can be too difficult to solve directly and instead we must resort to computers to give *numerical estimates* to our models. In some cases these can be near exact and others they can prove intractable. In this small workshop we will use Python, a scripting computer language, to illustrate how numerical modeling works and compare it against some familiar physical models.

Q1: What are some physics equations that you may be familiar with? What do they model?

2 Python scripting

Python is an *interpreted* language. This means that you can write a series of commands directly on a console, or typed in a script, and use Python to execute them.

Let's begin with some basic commands. Start python in your machine and try some of the following:

```
print("hello world")
```

```
10+7
```

```
3*(abs(-3)+-3)
```

So you see we can do basic arithmetic with direct numerical input. We can also define *variables* and achieve the same thing. Consider:

```
x=3; y=2; x*y+x-y
```

Before executing, what should the answer be? Did it agree with the output?

Now try some of the following:

```
sqrt(4)
```

```
log(10)
```

```
pi=3.14159; cos(pi)
```

Did it work? Why or why not?

2.1 Libraries

As you saw, Python comes with some limited default functions like `abs()`,`print()` etc. but also misses some other important functions. We can expand Python capabilities by using *libraries*. Libraries are, in short, just compendium of more advanced functions, operations and other procedures that allow you to immediately carry out more complex tasks depending on your needs.

For our purposes we will use the 'numpy' and matplotlib libraries. These will allow us to access more advanced mathematical functions and plotting capabilities.

Now try the following:

```
import numpy as np
np.sqrt(4)
np.log(10)
np.cos(np.pi)
```

Did it work? What is the '.' above for?

2.2 Arrays and Loops

In addition to variables, we also make use of *arrays*. These are essentially containers of individual variables of the same type (e.g. integers) which makes it easier to operate and keep track of in a code. For instance, consider the sum of two vectors in two dimensions:

$$\mathbf{a} = x_a\hat{\mathbf{i}} + y_a\hat{\mathbf{j}} \tag{1}$$

$$\mathbf{b} = x_b\hat{\mathbf{i}} + y_b\hat{\mathbf{j}} \tag{2}$$

$$\mathbf{a} + \mathbf{b} = (x_a + x_b)\hat{\mathbf{i}} + (y_a + y_b)\hat{\mathbf{j}} \tag{3}$$

These can be coded in as follows:

```
a=np.asarray([1,2])
b=np.asarray([3,-1])
print(a+b)
```

where we have chosen $x_a = 1, y_a = 2, x_b = 3, y_b = -1$.

Notice Python has automatically done the term-by-term addition. Python also allows direct accessing of a variable via bracket `[]` operators and index number starting from 0 for the first element. Thus we can read out the vector components:

```
print("x_a is ", a[0], "and y_b is", b[1])
```

Q2: What would the command to access y_a and x_b look like?

Suppose we have an n dimensional array. How can we declare it, set, and access some specific range of values? Use brackets!

```
#Note: anything following '#' is a comment; will not execute
n=10; #dimension
a=np.zeros((n)); #array of 0s
a[3:6]=2; #set from index 3 to 5!
print(a[1:4]) #what numbers will this print?
```

When n is too large, setting values by hand gets cumbersome. To get around this we can use a sequential loop, or iteration, which means we carry out a block of code a repeated number N of times. Try:

```
N=n; #recall n=10 from before
for i in range(N):
    a[i]=i;
    print(a[i]);
a[:]=0; #reset values for questions below
```

Q3: What happens if we let $N = n/2$? What values do you get if you let a) $a[i] = i * i$ b) $a[i+1] = a[i] + 1$ where $N = n - 1$?

2.3 Graphical plotting

Whenever you have data to analyze one of the easiest and most productive ways is to create graphics via plots. To this end we will use the 'matplotlib' library which includes all the necessary graphical tools. As an example, let's plot the functions $f(x) = x^2$ and $f(x) = \cos(x)$.

```
import matplotlib.pyplot as plt
dx=1e-3; x=np.arange(0,1,dx);
fx=x**2; fcos=np.cos(x);
#use matplotlib to create plots
plt.plot(x,fx,label='x^2') #plot 1
plt.plot(x,fcos,label='cos(x)') # plot 2
plt.xlabel('x');
plt.legend();
plt.show(); #show plot window
```

Q4: Can you explain the code above? Which of the above variables are a) arrays, b) constants.

3 Physics Modeling: Projectile Motion

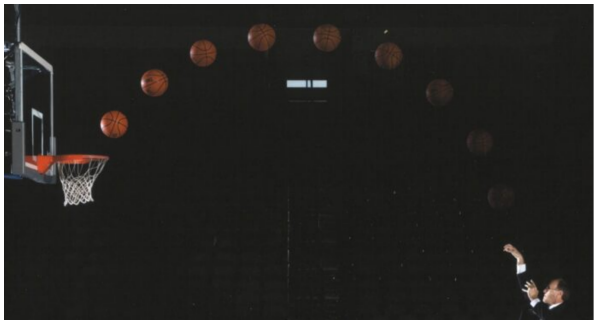


Figure 1: An object thrown near the surface of Earth follows approximately parabolic trajectories described by kinematic equations.

Projectile motion is the quintessential example of physics modeling. Following Newton's laws of motion we know that $\mathbf{F} = m\mathbf{a}$ where m is the mass and a is the acceleration. To determine the equations of motion we need to have knowledge of a particle's position $\mathbf{x}$ and its velocity $\mathbf{v}$. Solving these equations for constant acceleration, as is the case near the surface of the Earth, leads to the well known kinematic equations taught in every physics class.

For instance, you may recall that the vertical position y under constant acceleration g is given as a function of time t as

$$y(t) = -\frac{1}{2}gt^2 + v_0 t + y_0 \tag{4}$$

where v_0 and y_0 are the starting velocity and position at $t = 0$, and we chose the positive direction to point up (so gravity points downward). This expression is our *analytical* result from solving the Newton's equation of motion directly.

Q5: If we throw a ball straight up with $v_0 = 20\ m/s$, how long will it take to return to the point from where it was thrown i.e. $y - y_0 = 0$ for $t > 0$? Assume $g = 10\ m/s^2$.
In this part of the workshop we will now calculate the same one dimensional trajectory *numerically*. That is, we will assume we don't know the equation and instead calculate it from small continuous changes Δt up to some time t.

In particular, suppose we start from some velocity point $v_y(i)$ and position $y(i)$ where i is just an index along discrete time steps. Then for a small time interval Δt the new velocities and positions will be

$$\Delta v_y(i+1) = -g\Delta t + v_y(i) \tag{5}$$

and

$$\Delta y(i+1) = v_y(i)\Delta t + y(i) \tag{6}$$

respectively, where v_y is given by eq. 5 above.

The idea is then to start from a fixed value of $v_y(0) = v_0$ and $y(0) = y_0$ and iteratively evolve the system over $N = t/\Delta t$ steps which we do via a 'for' loop. This form of solving the

equation of motion is called an Euler-scheme.

Exercise: Now let's implement it via Python. Try to do the following steps:

1. Code the above scheme using $g = 10\ m/s^2$, $v_0 = 20\ m/s$, $\Delta t = 0.01s$ for $t = 4s$. Save the entire directory in an array of velocities and positions v_y and y respectively.

2. Plot the resulting trajectories for y and v_y and compare it for the analytical result for position in eq. 4 and $v_y(t) = -gt + v_0$. How does the accuracy of the numerical result compare with the analytical as we move forward in time?

3. Suppose we now also wish to compute the trajectory along the horizontal i.e. 'x' component. How would you adapt the code above to include the additional dimension?

3.1 Projectile motion with drag

Newton's kinematic equations above are reasonably accurate in the idealized case where energy is perfectly conserved and projectiles experience no added forces. However, real projectiles experience drag forces $\mathbf{f_d}$ as a result of moving through a medium like air. Exact expressions are usually system dependent (shape of ball, surface area etc.) but typically follow a relation of the form

$$f_d = -bv^n \tag{7}$$

where b is some coefficient that depends on the projectile properties, and n is an exponential power that account for the measured drag response with velocity. For our purposes we will take b to be variable and $n = 1$.

Following Newton's laws of motion, the added drag force means that the acceleration of the system is now described as

$$m\mathbf{a} = \mathbf{F_g} + \mathbf{f_d} \tag{8}$$

where $\mathbf{F_g}$ and $\mathbf{f_d}$ are the gravitational and drag forces respectively.

Focusing here only on the vertical component, we can solve the equations of motions analytically and can be shown that

$$y(t) = -\frac{mg}{b}t - \left(\frac{m^2 g}{b^2} + \frac{mv_0}{b}\right)(e^{-\frac{b}{m}t} - 1) + y_0 \tag{9}$$

where $e = 2.71828...$ is the exponential constant.

This expression looks nonetheless a bit complicated and for different cases of f_d, eq. 8 may not have analytical solutions at all. Thankfully, we can always evaluate it numerically and still be able to get a prediction for the trajectory of our projectile.

Exercise:

1. Modify the above code for the kinematic equation and include the drag force f_d. Assume $m = 1$ Kg, $b = 0.1$ Kg/s and solve for up to $t = 2s$ of trajectory time.

2. Compare the numerical solutions with the analytical equation.

3. Compare the dragged trajectory results with the ideal kinematic results. Do the results make sense? How should the trajectories differ due to the presence of the drag?