What is computational physics?

There are two main ways in which computers are used in physics. The first is for use in experiments for the measurement and analysis of data. The second is for performing numerical analysis on problems for which analytical solutions are prohibitively complex, chaotic, or computationally expensive. Our materials are concerned with this second use.

For many problems, especially in mechanics, the computational methods involved need not be especially complicated. The mathematics of simple differential equations is often much simpler to code than it is to analyze, meaning that high-level, accessible languages can be utilized to describe otherwise complicated problems.

The key choices involved in solving a physics problem computationally are found in how to represent the system discretely, as opposed to continuously. We must divide any dimensions in space and time into finite pieces that can be used in calculations. This often has a large bearing on both the time the problem will take to solve and the accuracy of that solution.

Why use computational methods in my physics classroom?

Recently, The National Research Council's *Framework for K-12 Science Education* proposed an increased focus on students' engagement in authentic scientific practices. In a rapidly changing world, where problems are increasingly complex, most scientists have employed computational methods as a basic tool for analyzing data and modeling phenomena. In this vein, the Next Generation Science Standards (NGSS), which have been widely adopted by many US states, identify computational thinking as a core scientific practice for students in grades 9-12. Students are expected to "analyze data using tools, technologies and/or models (e.g., computational, mathematical) in order to make valid and reliable scientific claims…" (HS-PS2-1, NGSS, 2013). Adopting computational methods in the physics classroom can offer students experience with a valuable scientific practice as well as allow students to engage with aspects of phenomena that they might not otherwise have access to.

Computational tools can serve as a powerful supplement to any physics curriculum by:

1. Allowing students to model or visualize phenomena that
    a. are too big or small to observe (i.e. microscopic phenomena, celestial motion, etc.)
    b. are too complex to observe (i.e. chaos theory)
    c. happen over large time scales
2. Providing a venue for students to utilize methods of discretization which:
    a. allows students to make sense of phenomena that normally require an understanding of complex mathematics.

   b. helps students develop a more sophisticated understanding of and intuitive grounding for calculus.

3. Allowing students to solve problems that do not have analytical solutions.

How to use these instructional materials:

There are 3 items in this instructional package:

1. A mathematica simulation of a ball with and without air resistance (air resistance is simulated in both x and y directions). Students can toggle values for an object's mass, the drag coefficient, initial speed, initial angle, and gravitational acceleration and then see how an object's motion changes over time. There are also thought questions provided in the mathematica notebook, which you can use or change to accomplish your instructional goals. There is also a template to help students analyze real-life motion. They can upload videos of dropping different objects, and compare the y(t) vs. t trajectories to the simulation graph.
2. A python instructional template designed to guide students through the process of coding their own air resistance simulation.
3. A fully-coded solution script. This script outputs a simulation of a bouncing ball with and without air resistance, and a graph of velocity vs. time for each ball.

   To run the Python script, type "python air_resistance_main.py" into a terminal window.

   a. To run the student-written code, the air_resistance_main.py should include this line:
      ```
      from air_resistance_funcs include *
      ```

   b. To run the fully-written solution, the air_resistance_main.py should **instead** include the line:
      ```
      from air_resistance_solution.py include *
      ```

Useful documentation:

Linux:

For instruction on how to install visual python:
http://vpython.org/contents/download_linux.html

(this download will also include a fairly robust text editor called VIDLE, which can be used to work on the python code)

Terminal commands to install visual python:

```
sudo apt-get install python-visual
sudo apt-get install libgtkglextmm-x11-1.2-dev
```

Terminal command to install numpy:

```
sudo apt-get install python-numpy
```

Windows:

For instruction on how to install visual python:
http://vpython.org/contents/download_windows.html
Install numpy

Mac:

For instruction on how to install visual python on mac:
http://vpython.org/contents/download_mac.html

in terminal:

```
sudo apt-get install python-visual
sudo apt-get install libgtkglextmm-x11-1.2-dev
sudo apt-get install python-numpy
sudo apt-get install python-scipy
```

Raspberry Pi:

To open raspberry pi on desktop mode:

```
login name: pi
password: raspberry
then enter: startx
```