Comptuational physics

Motivations for learning computing as a physicist

There are a variety of reasons why anyone completing a physics degree should develop a certain level of skill at computer programming. The two main academic reasons are to be able to solve problems that simply cannot be solved with just paper and pencil, and to be able to work with the large data sets that are becoming increasingly more common in most areas of physics. The other major reason, of course, is that people who know physics and can do at least at little bit of computer programming are in high demand in industry.

Numerical solutions to physics problems Many of you may have already heard terms like *closed-form* or *analytic* solution in physics classes. A function is said to be written in closed form if it has been expressed in terms of a finite number of "well-known" operations that lend themselves to calculation. "Well-known" has different meanings, but generally means the functions that you learned before you took your first calculus class – addition, subtraction, multiplication, division, logarithms, roots, exponents, and trigonometric functions and their inverses. The term *analytic* also includes some solutions to different kinds of differential equations – these are functions which most mathematicians and physicists will have heard of, and which may pop up if you try to get an answer to something from e.g. Wolfram Alpha, but which will not be represented by a button on a $20 calculator.

Whenever a solution can be expressed in closed-form, this is ideal. One can then easily make plots of the functional form of the solution and develop an intuition about how properties change with respect to one another. For example, when I write that:

$$y = y_0 + v_{y,0}t - \frac{1}{2}gt^2 \tag{1}$$

$$x = x_0 + v_{x,0}t \tag{2}$$

many of you immediately realize that whatever object we're talking about follows a parabola in space, and if I told you to prove that, by finding an expression for y in terms of x, you could solve equation 2 for $t = \frac{x-x_0}{v_{x,0}}$, and substitute into equation 1. This is the power of a close-form solution – you can integrate it into other work very easily.

Historically, it was quite useful to get equations expressed in analytic form where closed-form solutions were not possible. The reason for this is that there were books that tabulated the values of special functions like the Bessel function or the error function (which is the integral of a Gaussian, something we will come back to later in the course). One could then compute the numerical values of analytic form solutions to problems much more readily than one can compute the value of a non-descript, unsolvable integral.

In recent years, numerical solutions have become much more prominent in physics. When we say we make a numerical solution to a problem, what we mean is that we go back to the origins of calculus. Instead of using some pre-set methods for solving integrals, we treat the integral as what it is – the area under

a curve. In the simplest approach, we put a bunch of little rectangles along the function and add up the areas of the rectangles. Often other shapes will give much better answers with fewer shapes, but we will get to that later in the course.

The use of numerical solutions also us to re-visit some "classic" physics problems. Before the advent of computers, physicists would often "cheat" to solve problems by making approximations. A famous example is the small angle approximation, in which $\sin\theta$ or $\tan\theta$ is set to be equal to θ for values of $\theta << 1$ (remember: the angle must be in radians!). We can also add in air resistance to our projectile problems, for example.

Now, an important point to bear in mind is this: being able to program a computer in order to do a numerical problem does not absolve you from having to learn math. There are several major reasons for this, even looking ahead as a professional physicist (i.e. apart from your grades in classes). First, remember that the closed form solutions to problems are always preferrable to any other class of solution, because it's so much easier to work with them. Second, in many cases, if you do some clever algebra before you write your computer program, you will be able to set up a program that takes many fewer cycles on the computer's processor to run. In some cases, you'll want to use these advantages to get better precision, and in others just to get the job done faster. Third, you always want to be able to come up with some quick analytic expression to check your computer code against. Coding mistakes are far more common than algebra mistakes.

There are, of course, cases where our computers are still not powerful enough to do everything we want to do, and so we have to fall back on classical approaches, or, in many cases, we need both to make approximations *and* use the computer to solve the approximate equations.

Big data

In many fields of science, data sets are growing in size literally expoentially, and the data sets in these fields are almost entirely digital. This has made it both necessarily, and straightforward to analyze the data on computers, often with considerably less human intervention than was the norm in the recent past.

Just to give some examples from my own field of astronomy: a large number of projects now focus on detecting transient sources of light in the sky. These may be supernova explosions, fast moving asteroids, outbursts from black holes sucking in gas from their surroundings or any number of other things.

As recently as the mid-1990's, surveys of large parts of the sky were still done using photographic plates – basically something like old-fashioned camera film, in which light interacts with silver causing chemical reactions to allow detections of light. After being exposed on a telescope, the plates then needed to be developed chemically.

Now, the data for projects done on optical telescopes are generally collected using charged coupled devices (CCDs). These are digital detectors, which immediately convert the photons they detect into electrons, which can then be counted to measure the brightnesses of the objects being examined. The original camera for the Hubble Space Telescope (which was launched in 1990) contained

an array of 4 800×800 pixel chips (i.e. about 2.5 Megapixels). More recent projects make use of cameras with many Gigapixels. The data flow rate from radio telescopes has increased similarly – the Very Large Array in New Mexico[1], for example, is in the process of having its data rate increase by a factor of about 20, as its electronics are being upgraded from the original 1970's era electronics to modern equipment.

At the present stage of transient search experiments, the detection of an event that is a transient must be made by some automated computer routine. There are simply too many images being taken every night, with too many stars in each image, to have a human look at the data and see which objects have varied. At the present time, the total number of *transients* is still small enough that humans can examine the data to look at them. However, as even bigger projects are built, this, too may become difficult, and people are working on writing computer programs that would classify the transients based on the data. Similar types of advances in data processing are going in most branches of physics and, indeed, branches of all sorts of other science and engineering fields.

What we will do in the course

So, having set some motivations out for doing a course in computational physics as part of a standard physics degree, let's set out the actual goals of the course.

Working in a Unix-based environment

There are a variety of operating systems for computers. Most of you are probably most familiar with using Windows or MacOS. MacOS is a Unix-based operating system, although many people who use it do not take advantage of its power.

The advantages of operating systems like Windows are also its disadvantages. Most things are done by point-and-click. This approach makes it easier for the user performing simple tasks to get started, but makes it more difficult to do sophisticated things. In Unix, one can open a "shell" window, and interact at a more basic level with files on the computer in a more straightforward manner. This makes Unix the operating system of choice for a lot of the things that most physicists do – if, for example, you want to perform the same analysis on a large number of different data files. It is also generally true that there is a better suite of free software available for Unix systems than for Windows. This difference ranges from being mildly important to absolutely crucial depending on one's field of endeavor.

Plotting data

Graphical representation of data is a vital tool to (1) understanding what's happening in physics and (2) conveying that to other people. Throughtout the course, you will plot your data. Part of your grade will be based on choosing good ways to present your results. For example, if you want to show that two quantities are related by a power law function, plotting the logarithms of

[1]This is the big Y-shaped set of 27 radio dishes that you may remember from the movie Contact.

both will generally be more illuminating than making linear plots of both. For exponential relations, you will want to take the log of only one quantity.

Understanding some basic numerical methods

Much of what you will do in this course will involve going back to the roots of calculus and just adding up numbers repeatedly in order to do integrals. However, there are schemes which can be far more efficient than others for doing this sort of things. There are also cases where setting up the equations incorrectly on the computer can lead to big errors, because the computer rounds numbers off.

Some beginning of understanding error analysis and model fitting

We will look at how computers can be used to estimate parameter values from experimental data. For example, suppose you tossed a ball in the air, and measured its height a large number of times. How would you know (1) what g is? And (2) what the initial velocity was with which you threw the ball? And how do you take into account the uncertainties on the measurements that you made? You can do this using a computer quite easily, and it is quite hard to do *in a precise, systematic manner* by hand.[2]

Understanding the process of writing a computer program – and that it's not that hard

This is probably the most important thing to do in this course. As computers have become more sophisticated, and software has become one of the biggest sectors of society, tech-savvy students are often much *more* intimidated by computer programming. The things that computers can do are much more sophisticated now than they were 20 years ago when I was taking courses like this one. Richard Feynman once said to the effect that people think physics is difficult because they try to start at the end – they try to understand quantum gravity without understanding how to solve pulley problems. Computer programming can appear difficult in the same manner.

[2]You can, of course, get a decent guess by hand.