Journal of Physics: Conference Series **759** (2016) 012089 doi:10.1088/1742-6596/759/1/012089

Open Source Computing in Physics Education

Ananda Dasgupta

Indian Institute of Science Education and Research Kolkata, Mohanpur Campus, India - 741252

E-mail: adg@iiserkol.ac.in

1. Introduction

No one can deny that computers are important tools for a physicist. The aspect that we are most familiar with is that they are great for solving research problems! What is not so widely appreciated is that they are also great as teaching tools. They are grossly underused in this aspect. In this talk, I will try to demonstrate how an instructor can make use of the computer to make her course both more attractive and effective . The focus of the lecture was on three aspects

- writing your own animations for physics teaching.
- teaching students to explore physics through computing.
- using open source software for simple experiments.

For the sake of brevity, I will discuss only the first one in this article. In this, I will assume a reasonable amount of computational skill on the part of the instructor. However, one does not have to be an expert in computational physics in order to make use of the methods described here.

2. Animations as a teaching tool

All teachers will agree that animations can be very useful for improving student comprehension of complicated topics. They can serve as complements for traditional laboratory experiments, as well as aids for thought experiments allowing students to explore parameter regimes that might to too tedious or too expensive otherwise. Given their potential, it is natural that a lot of work has already been done on this topic. A huge repository of animations dealing with various aspects of physics are already available. This includes java applets, shockwave animations, vpython animations and a lot, lot more. Some of these are commercial, while a lot is available for free on the internet. It needs to be stressed that some of the free software is of extremely high quality - often they are superior to their commercial counterparts.

Despite the easy availability of commercial and non-commercial software there are several stumbling blocks on the road to adapting the available ones for the classroom. Even if you manage to find one that deals with the subject at hand, the available software may not be exactly suitable for the purpose at hand. At the very least, you may want to use other values for the parameters which are hard-coded. What's even more likely is that the software may not deal with the exact aspect of the problem that you want to explore. The best solution to this problem is to write your own animations!

This solution comes with its own set of problems though. After all, it takes time and effort to learn a programming language! Given the busy schedule most educators have it would be

too much to expect this from them. Even if one is used to programming for solving research problems it takes a lot of time and effort to write animations. A way forward that I propose here is to make use of the animation capabilities built into **gnuplot**.

3. Gnuplot

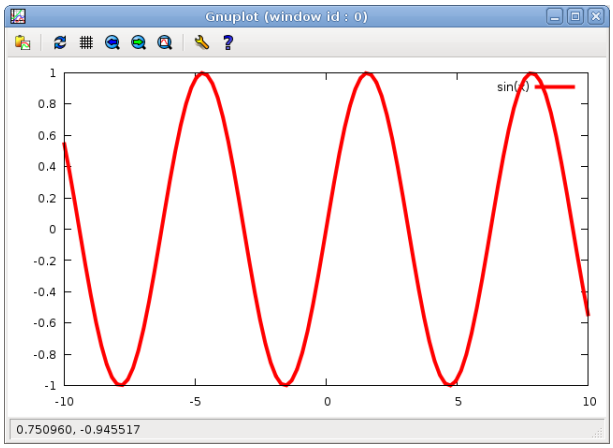
There are several reason why we recommend gnuplot. To begin with, the best reason is that it is open source. It is available free of cost - this may be an important consideration for many of us. Given the wide variety of operating systems that are in use, another plus point is that versions of gnuplot is available for most of them. The most important consideration is that it is relatively easy to create high quality plots using this software and what's even more important for the present discussion - animations are also easy to create!

3.1. Plotting with gnuplot

By default, gnuplot is used for creating 2D and 3D plots. For example, the command

```
plot sin(x)
```

produces the following graph

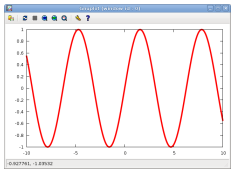
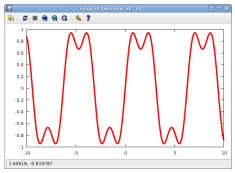
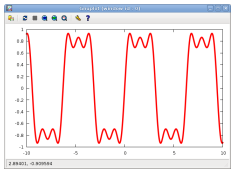


As you can see, by default gnuplot takes the independent variable to run from -10 to +10. This and many other aspects can be customized. For details, please refer to the gnuplot documentation.

We will be more interested in gnuplot scripts. At the most basic level, a script is nothing but a sequence of gnuplot commands written in a file. for example, consider a file with the following lines

```
set sample 1000
plot sin(x)
pause 3
plot sin(x) + sin(3*x)/3
pause 3
plot sin(x) + sin(3*x)/3 + sin(5*x)/5
pause -1
```

will display the graphs shown below

after intervals of 3 seconds (that's what the 'pause 3' are for). By default gnuplot plots a curve by plotting 100 equally spaced points. The first line in the script changes this number of points to 1000 - leading to a smoother curve. The final pause -1 keeps the pauses the program indefinitely (that is until the user hits a key) so that the picture persists on the screen.

3.2. The reread command

When gnuplot encounters the **reread** command in a script it loads the script again. This is the basic step that allows us to write animations in gnuplot. To see how this works take a look at the script 'easyWave.plt' containing the following

```
# file 'easyWave.plt'
# Plots a running wave

plot sin(x-ct)
pause sp
ct = ct + sp
if (ct<5) reread
```

To run this script open the gnuplot interpreter and enter the commands

```
ct = 0
sp = 0.05
load 'easyWave.plt'
```

and you will see a wave running across your screen.

If you want to run the script directly without opening the interpreter, you must put the three lines above in another script **initWave.plt** . Running this script by typing **gnuplot initWave.plt** at the command line (for linux) or double-clicking the icon (in Windows) will give you the running wave.

3.3. More complex examples - Fourier series again

The Fourier series for the square wave

$$\mathrm{sq}(x) = \sum_{i=0}^{\infty} \frac{\sin[(2i+1)x]}{2i+1}$$

leads to the partial sums

$$\mathrm{sq}_n(x) = \sum_{i=0}^{n} \frac{\sin[(2i+1)x]}{2i+1}$$

which obey

$$\mathrm{sq}_n(x) = \mathrm{sq}_{n-1}(x) + \frac{\sin[(2n+1)x]}{2n+1}$$

We can use this recursively, along with $\mathrm{sq}_0(x) = \sin(x)$, to calculate $\mathrm{sq}_n(x)$

To create an animation that will show the partial fourier sums for the square wave in succession, we need the initialization script **initFourier.plt** :

```
# initFourier.plt
# Fourier series for the square wave

sq(x,n) = n == 0? sin(x):sin((2*n+1)*x)/(2*n+1)+sq(x,n-1)
```

```
set sample 1000

n = 0
load 'Fourier.plt'
```

and the main script **Fourier.plt**

```
t = sprintf('Fourier Series summed to %d terms',n+1)
set title t font 'Helvetica,16'

p [-10:10] [-1.5:1.5] 4/pi*sq(x,n) t 'square'

n = n+1

pause -1 "Hit enter to continue"

if (n<10) reread
pause -1 "Hit enter to quit"}
```

Running this script by typing

`gnuplot initFourier.plt` will result in the display of the first ten partial sums of the Fourier expansion of a square wave, with gnuplot waiting each time before moving on to the next graph for the user to strike a key.

4. Conclusion

In conclusion let me emphasize that it is quite easy to master the art of creating high quality, flexible animations using gnuplot, which can go a long way towards aiding the teaching-learning process.

[1] The gnuplot website *http://www.gnuplot.info*.