Explain the color wave movie demo

This file explains the Python code `colorWaveMovie.py`. If you download and run this script, it makes a file `colorWaveMovie.mp4`. Depending on your computer, you might appreciate how Python graphics got its reputation for slowness.

I confess that I do not understand everything about this code. The documentation is not as helpful to me as the basic Python documentation. I would be glad for any explanations from people who know more. In particularly, it's likely that some stuff here is done in a dumb and possibly buggy way. Please let me know.

The code visualizes a function $v(\theta, \phi)$ depending on an amplitude A, a frequency ω, and a decay rate λ:

$$v(\theta, \phi) = A\cos(\theta - \omega\phi)e^{-\lambda\phi} \ .$$

Think of θ as the "variable" and ϕ as a "parameter". The movie visualizes the behavior of the curve $v(\cdot, \phi)$ as ϕ changes. Each frame visualizes v as a function of θ for a fixed ϕ. Watching the movie tells you how this function depends on ϕ. This notation is used in Assignment 5, but with different functions. The function is defined on a curve

$$y = \sin(\theta) \ . \tag{1}$$

The function v is visualized through colors of points on (1). In the assignment, the curve (1) is replaced by one or more ellipses. The functions being visualized are the values v_k or possibly u_k. The angle ϕ represents the angle of the overall source point. Up to line 47, the code just computes the curve and the function values to be visualized on the curve.

Lines 49 and 50 compute the overall min and max of the data to be visualized. Without this, each frame would have its own scaling, which is a common visualization mistake, and you would not be able to see changes in the range of the data from frame to frame. You can see the decay in the movie, in the beginning you see the whole range of colors in the colormap. By the end you see a smaller range of colors because of the exponential decay of the amplitude of the oscillation.

Line 52 starts the visualization. Line 54 sets up the colormap. A *colormap* is a function, $C(x)$, (a "mapping") $color = C(x)$. Here x is a (floating point) number and C is a color specified in some way. In Python, a *colormap* is an instance of the `colormap` class. Line 54 instantiates such an object and binds the name `cmap` to it.[1] I chose colormap called `hot`. The matplotlib documentation

[1]The notes *Python for Smarties* explain what it means to *bind* a *name* to an *object* and what it means to *instantiate* an object of a given class.

on colormaps lists many others. *I do not understand how colormaps work.* Lines 59 to 62 set up the geometry of the plot image, with the plot figure on the left and a colorbar on the right. The colorbar reveals the mapping from numbers to colors that this colormap uses. The plots are made by the plot function `scatter`, which is used for *scatterplots*.[2] I essentially copy/pasted the `fig.colorbar()` command (starting on line 60) so I can't explain it in detail. Lines 64 to 68 and line 80 add to the plots in ways we have seen in earlier assignments. Standards of visualization apply to all visualization, not only simple plots.

There are two ways to make a moving image with matplotlib. One is *animation*, in which the moving image is created in real time by a running Python program. This demo uses the other method, which puts the individual frame images into a movie file that can be saved and played by a movie player. Line 74 instantiates the object that collects the individual images and creates the movie file that contains them. The `with` command creates a local *context manager* (another concept I don't understand well) to collect the images. One of the points of a context manager is the "cleanup" phase at the end. Line 86 is the end of the managed context region. When the interpreter gets to this line it closes the managed context using code contained in the context manager. In this case, that "closeout" code reformats the images it has collected puts them all into the movie that it writes to a file. Lines 79 to 83 create the images for the individual frames. Line 85 sends the current state of the `fig` object (the plot) to the `writer` object that was instantiated using the `with` on line 76. Notice that `fig` is the first argument to the `constructor` on line 76 that instantiates the `writer` object. That's how the call to the `grab_frame` *attribute* of the `writer` object on line 85 gets access to the plot. The second argument to the constructor on line 76 is the name of the file where the movie is to be saved. This happens (to repeat what I said above) before interpreting line 86 and is done by the closeout part of the context manager.

[2] A *scatterplot* is a bunch of dots in the plane, Each dot represents a data point. They are supposed to make a "scatter", but in this use they lie on a smooth curve.