This document contains descriptions of the functions and examples of how to run the code and what you should get.

A couple of things to note are that the E and B field are both calculated with Gaussian units which keeps them nearer to unity and easier to simulate.

There is no default plot type, but default angular frequency is 1 and default angular momentum is 2.

**runfunctions.py**

**besselPlot**

The bessel function plot is just a 2D plot which shows whatever bessel function you pick to simulate, it's there just to verify the radial term. The actual bessel function gets called from the functions.py file and is the summation of a handful of things. Interestingly enough though, the width of the Bessel function is dependent on the K\_r term which is determined by the kratio at the top of the runfunctions.py file. The square of the sums of the squares of the kratio array needs to be equal to one to make physical sense because K\_r and K\_z are the legs of a triangle with hypotenuse (omega/c) in k space.

**showEz**

This is one of the first attempts that I made to make an animation of the electric field in the z-direction. It doesn't necessarily need to be the e-field, we could go in and change to any of the other functions that we calculate. Given that we import the function into runfunctions.py and swap out the z = np.real(E\_z(... part in the drawE\_z(i) function. This one gives us the ability to play with textures but I didn't really like it all that much.

**slider**

This one shows the same thing as showEz, but it allows you to move a slider back and forth rather than having an automated setup. There is a handy command for buttons that I am trying to implement, where you can swap from one slider to another. So I'm thinking of incorporating a button to do the Ez and some other E or B components, I just don't really know on what axes those should be or how that transition would look.

**avg\_energy\_density**

This one is the plot of the time averaged energy density, as it is named. Pretty straight forward heat map, the interesting things come when you change the k\_r, k\_z, and omega terms. I still need to think about how to get the phase into this.

**functions.py**

**Bessel**

This solves for the bessel function by currently summing 25 different terms, ideally it should be infinity but this is the function that slows down the program the most. We need to consider tricks for optimization in here if we really start bogging down the computer. It takes into account the ability for the beam to spread out in the radial direction with the k\_r term and also solves for the radial distance from the azimuthal axis. It could potentially be made faster by solving for the radial distance somewhere else.

**E\_r**

The radial component of the electric field. In Gaussian units again, it solves for the angle from the x-axis each time and there is an if statement for when l = 0 because that is the only time when the derivative of the function will result in a negative Bessel function.

**E\_phi**

The phi component of the electric field, it has an interesting term in the denominator which is r\*k\_r which takes away all of the units and gives us Gaussian stuff again which is nice for computation.

**E\_z**

You know what this is. This is the big important one. Look how beautiful it is. That damn phi calculation makes it just slightly unattractive, I'll have to find a better place for it.

I'm not even going to talk about the magnetic field components because they are basically the same as the electric field components but the term with the derivative of the Bessel function swaps and is in the phi component so I need that if statement again to deal with l=0 Bessel functions.

**energy\_density**

Taken straight out of the Bessel Kirk paper on the drive. I just implemented it and figured I could solve for it later to add to the appendix

**Running the Code:**

To run the code all you have to do is save both the runfunctions.py and functions.py files in the same directory (Assuming you have the proper python libraries).

Open up a command prompt and type in the command,

python runfunctions.py -p [plot type] – l [int] -w [float]

The plot names are,

1. bessel

2. intensity

3. slider

4. showez

It shouldn't matter if you enter with uppercase, if it does that's a bug that I want to fix.

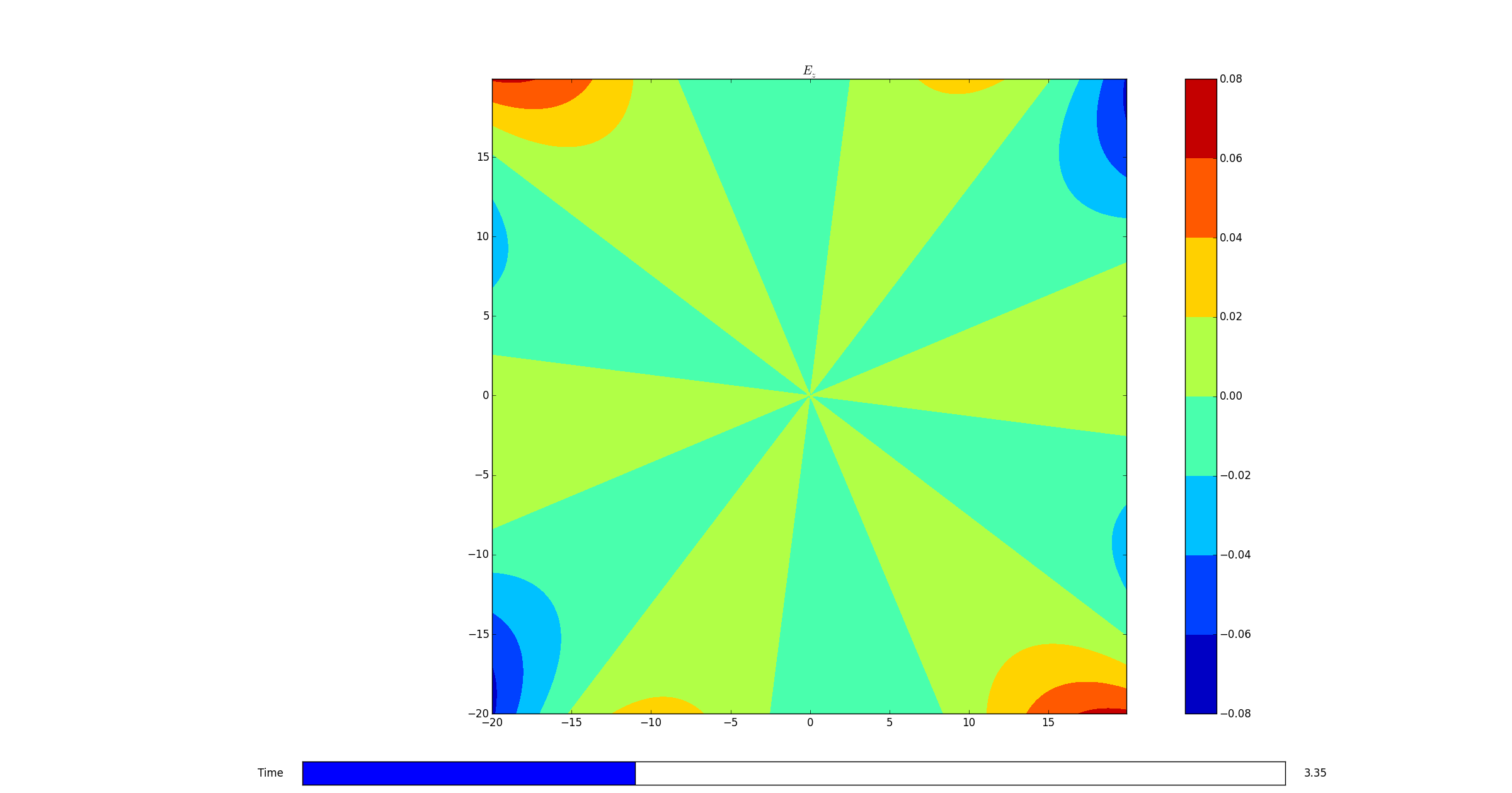
The -l term is the angular momentum term as you may have guessed and should be an integer, I haven't tried any fractional values in there so that may be a way to test for more bugs.

The -w term is the angular frequency in Gaussian units. For some reason it doesn't work when omega is greater than c, where c = 1. But between 0 and 1 the angular frequency works great. That means that there is more than likely a bug in here, but I'm too tired to really go and look for it right now.

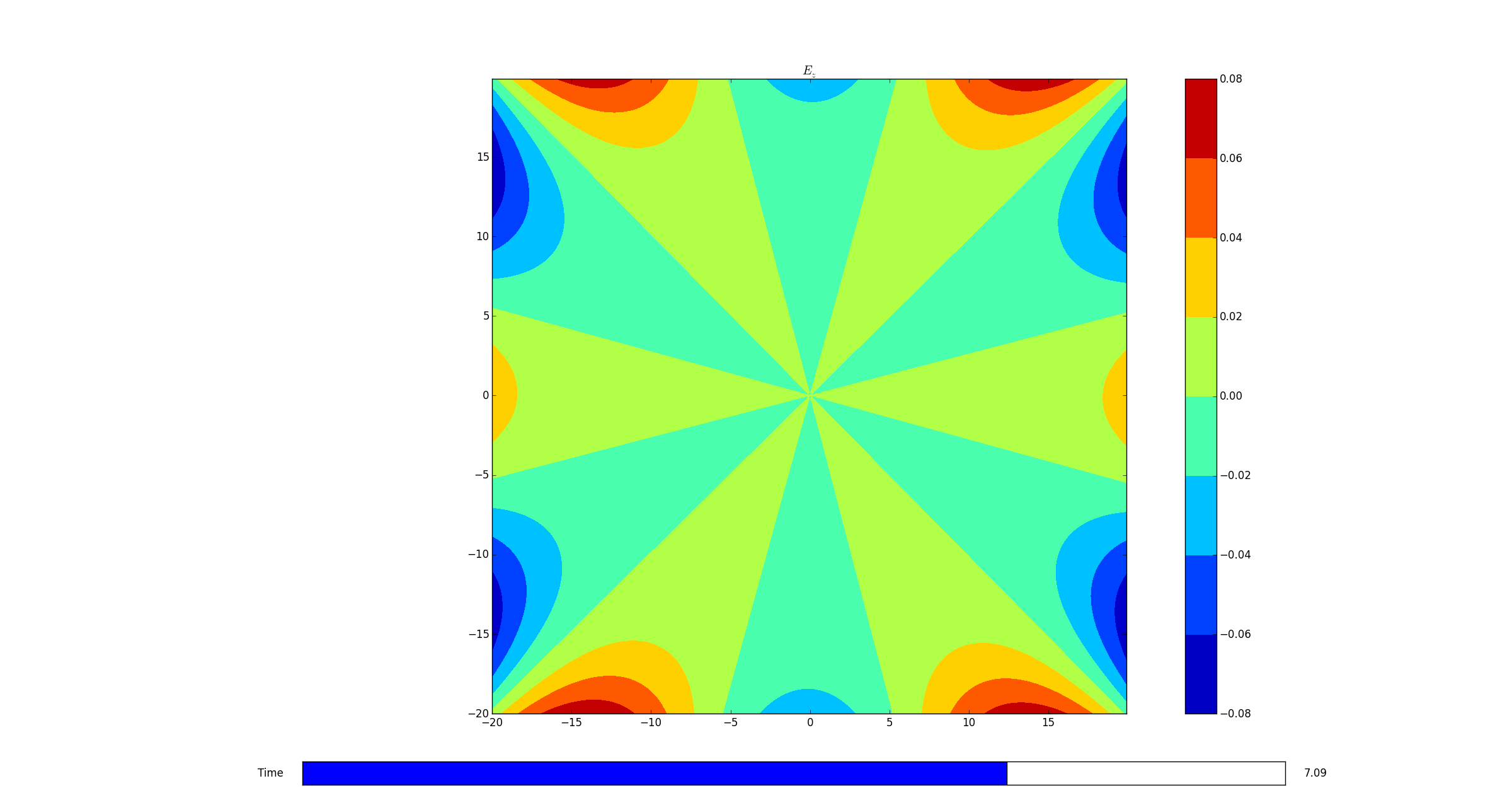
I give an example,

python runfunctions.py -p slider -l 6 -w 0.23

Which should give you an interesting design.

****

Example of What I see when I type in that command

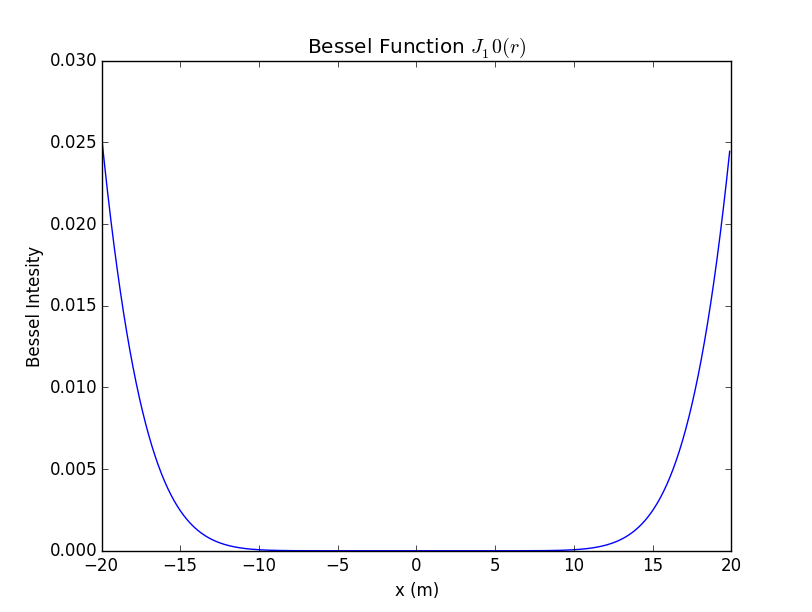
****

Another command that you could try is,

python -p bessel -l 10

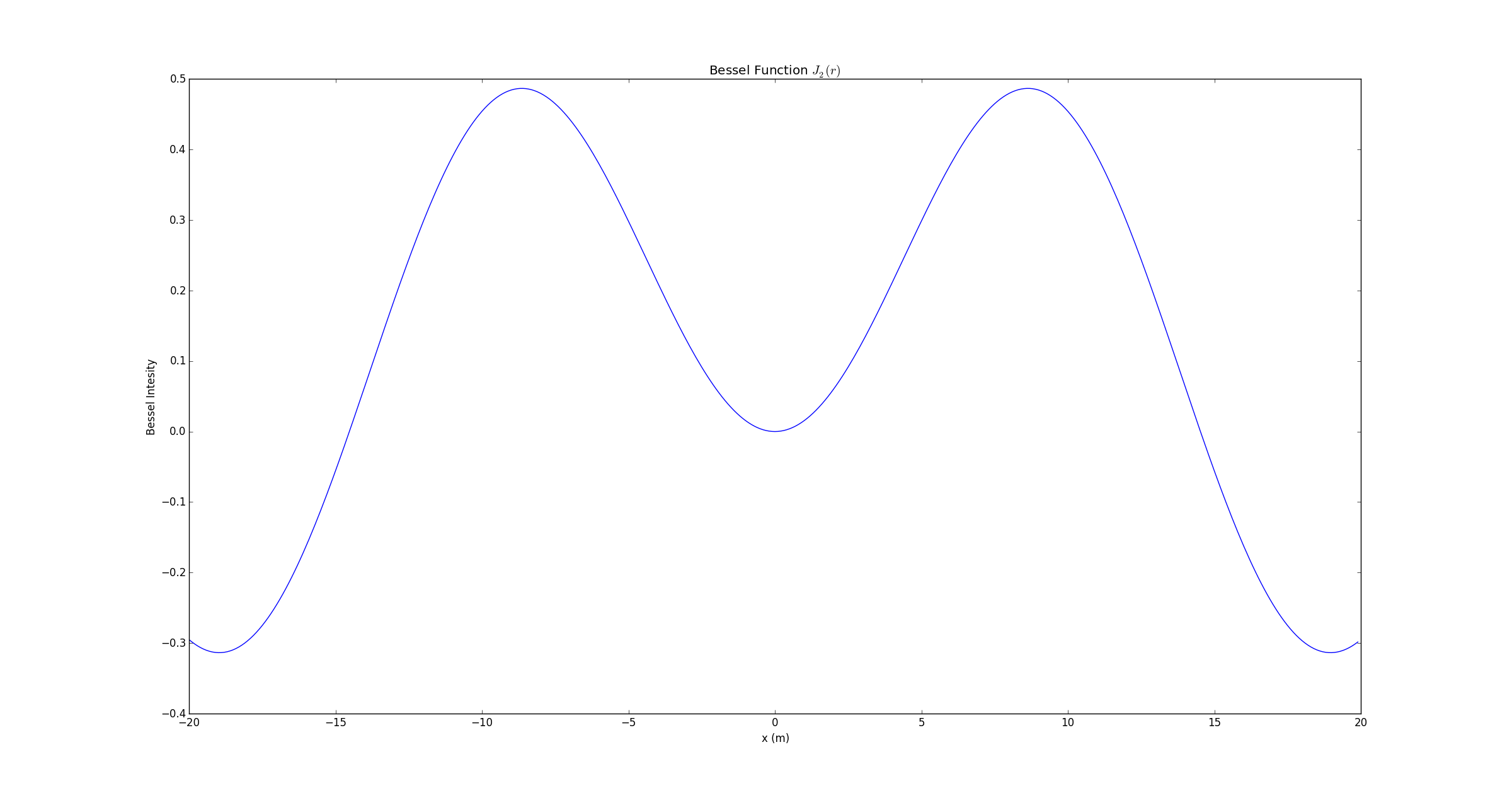
Which gives you the example that you don't need to type in all of the commands each time.

You could technically just type in a plot type and it will spit out the default type which are -l = 2 and -w = 1

Also try,

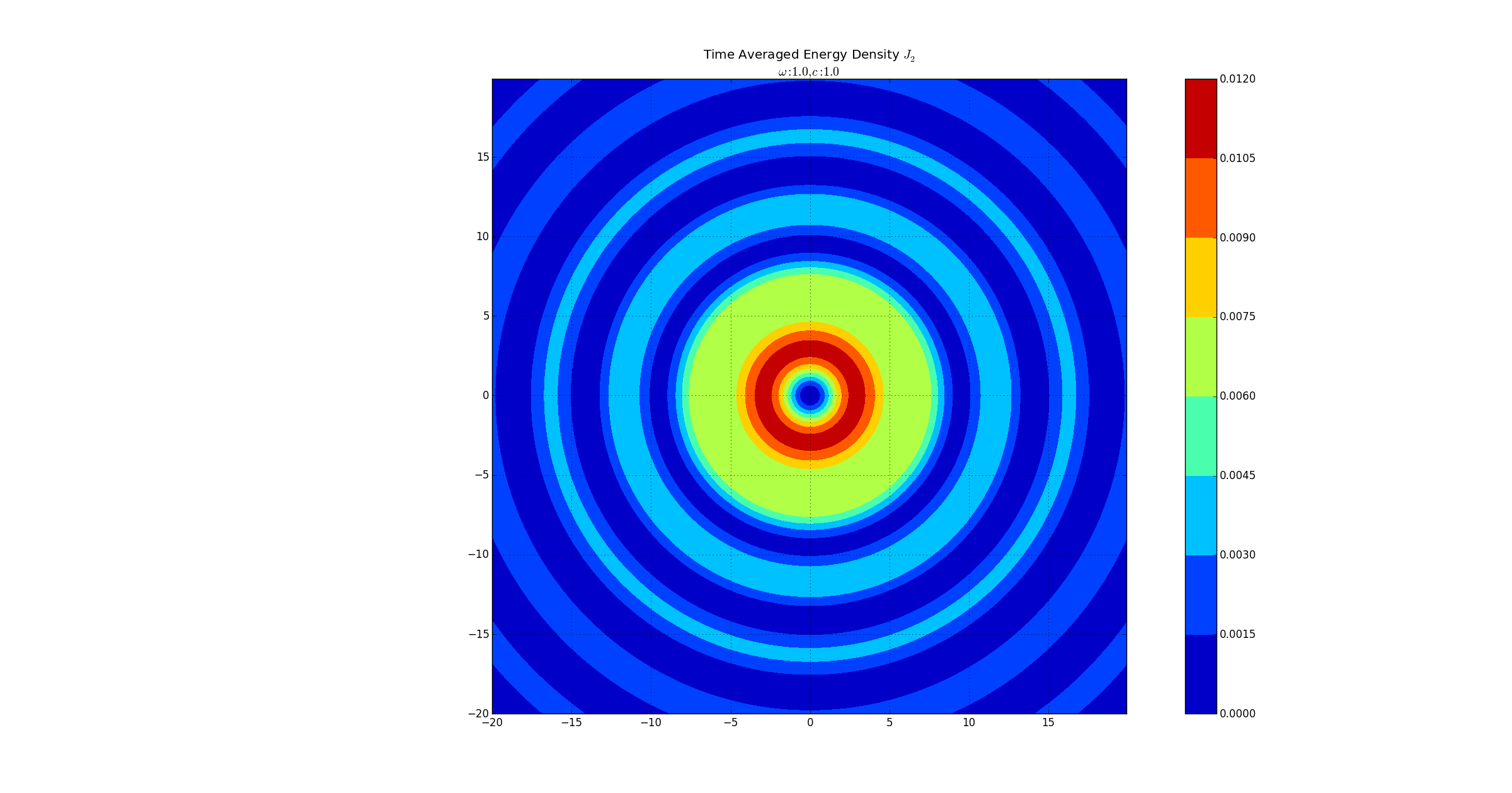
python -p bessel

and you should see our beloved bessel function

I don't know why you can't really read that.

I'll give one last example and what I see so that you guys can start playing with this a bit.

python runfunctions.py -p intensity

Awesome. 

There are a ton more things that we can do with this and I plan on focusing on this after thanksgiving.

I hope you read this, lol.