Package Reference

Physics 91SI Spring 2013

SymPy: A Python Library for Symbolic Mathematics

I used the SymPy packages which has a greater, more complex, array of mathematical functionality than NumPy. It adds the ability to do calculus and algebra with functions in Python. I used it for its Taylor term functionality. It is cool because it can do computer algebra. The setback had with it is its integration with matplotlib. I had trouble plotting and instead of using matplotlib I used SymPy's old plotting technique using Pyglet and therefore I downloaded Pyglet.

–Alex Zannos

Spyder: A Scientific Python IDE

We downloaded Spyder which is an interface and compiler for Python. It was a great tool because we were able to run our code really quickly and track mistakes and also search up different attributes and functions.

–Clara Warden and Sara Sheffels

animation: A matplotlib Submodule for Animating Plots

While looking at ways to plot the solar system, I came across this cool demonstration of animation, a package from matplotlib: http://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/.

Animation is particularly useful when you have data that is constantly being updated and may or may not have changed. To use it, you supply two functions, one (`init`) that has the base canvas of what your data should look like and a function (`animate`) that gets called between each successive frame. A neat feature about this separation of control is that the `animate` function takes in a variable that represents something like `time_elapsed`. In addition to those two functions, you give it basic information like the plot figure, the number of frames you want, the time interval to wait between frames, and an optional flag that saves redundant drawing when certain data hasn't changed.

Unfortunately, I wasn't able to implement this in my program, but I thought it was super cool and

wanted to share.

Also, animation provides an easy way to save the animations you've produced, a one-liner!

–Vincent Su

`scipy.optimize.curve_fit`: Easy Curve-fitting with SciPy

For my project, I worked extensively with the SciPy package. Specifically, I used the function `scipy.optimize.curve_fit`. This function takes 3 arguments: a function to fit to, a list of "x-values," and a list of "y-values." In the context of my project, I used the function to take a number of numerically generated accelerations varying from our model of the Newtonian gravity of the sun at different distances throughout the solar system and fitting them to a function that describes a possible non-Newtonian acceleration. The idea is that in this way we can "discover" a new force or increase our current understanding of the nature of gravity in deep space.

This function is particularly cool because of how simplistic it is, but also for how profound the results are. The `scipy.optimize.curve_fit` function actually uses a least squares regression to find the most optimal values for the unknown parameters in the given function. Also, the function operates surprisingly fast. I can implement a thousand iterations of this function in less than a minute (while doing a thousand iterations of my regular code also!).

I did have two main setbacks using this code. The first one is that if the function you are fitting to has any special mathematics (i.e. roots, powers, etc.) you have to be careful that the sequence you input supports those. For example, the list does not support the ** operator, so I had to change the sequence into a numpy array for it to work successfully. Second, the function sometimes has a problem with fitting several different parameters, in my case, returning the value 1.0 for the unknown parameters. I solved this by fitting to a single parameter a few times in order to get a reasonable answer. I also ended up solving this by inserting "good guesses" for the parameters into the function (which supports this).

–Brandon Buscaino

`os.listdir`: Exploring Directory Contents from inside Python

The name of the feature that I would like to write about is `os.listdir()`, which is implemented in the `os` package. The documentation for `os.listdir()` is...

```
    Return a list containing the names of the entries in the directory given
```

```
by path. The list is in arbitrary order. It does not include the special
entries '.' and '..' even if they are present in the directory.
```

which is very straightforward. This function was extremely useful to me for my project. The big challenge for my project was working through the numerous data files for each halo simulation area and extracting the right information from each one. I needed to have a list of all the data files that I could always access in my program, and `os.listdir()` provided a very easy way for me to get this. One quick note that I would like to make is that the `os` package provides another very useful function that pairs quite well with `os.listdir()`, and that is `os.path.isfile("path")`. This function returns `True` if the given path is an existing regular file. In the specific case of my program, I wanted only data files, so I could have used `os.path.isfile()` to filter out the non-files from `os.listdir()`. However, this was unnecessary in this case because the directory that I was getting the data files from only contained the data files. However, it was very useful for me to find the combination of these two functions because I can see how well they work together.

–Marc Williamson

`os.system`: Executing Shell Commands from inside Python

The `os` package (`import os`) is a pretty versatile package with many functions. For this project I focused on using the `os.system` module which allows for the use of UNIX shell commands to be executed within Python scripts as well as the `os.path` module which allows for file path manipulation to be done within Python.

Using `os.system("arg")` allows you to type in a standard UNIX shell command and have the Python script execute it. This is particularly helpful for my project in using `os.system('mkdir')` to create folders. Also the `os.path` module was useful as it has the ability to access system file paths within Python and then save these paths as strings for easy path manipulation later on. I used the `realpath()` function to get system address for my Python script and then also used the `dirname()` function to get the system path to the directory I wanted to create my input files in.

As my project was in file generation and manipulation, the use of these `os.system` and `os.path` packages was really useful as it enabled me to write a script to generate 100+ distinctive files within properly sorted locations which would be extremely time consuming to do manually through the Linux shell.

–Sunil Deolalikar

`numpy.genfromtxt`: Data File Loading with NumPy

One of the most useful functions that I found for my project was `genfromtxt`, from NumPy. `genfromtxt()` is described as a function that loads data from a text file and can simply take a filename as an argument. `genfromtxt()` would then return an array of the data points. What made the function useful, however, was its flexibility with different forms of files. For example, for my IV data files, I would often have extraneous information at the beginning of the file before I got to the data points. I called genfromtxt with the following line:

```
genfromtxt(filename, skip_header = 25, usecols = (0, 1))
```

This allowed me to ditch my working, but messy and confusing, regex and focus on cleaner, more understandable code.

Doing so allowed me to skip this extra information and also allowed me to specify what columns I wanted to use. The function also has many other optional arguments that allows you to specify other qualities.

One small setback that I had was the inability to use this function while loading my module in IPython on a corn server. It just came down to simply importing `genfromtxt` from NumPy. From this, I learned that you need to import all required methods within the module code!

–David Lam

`scipy.optimize.fmin`: Minimizing a Function with SciPy

This feature is used to minimize a function by repeatedly adjusting a parameter that you pass into it. It takes as arguments the function to be minimized and your initial guess as to what parameter value will yield the lowest function value, and returns the parameter value that yielded the lowest function value. Your function should have the parameter as its only argument. I used this feature to minimize the total distances of a point from each member of a set of lines.

This feature is cool because it is very straightforward to use but still quite powerful, in that it can be used on basically any function that has a minimum. I didn't have any setbacks with this feature thanks to Rex showing me how to use it, but I did have a lot of trouble initially because I tried to use `leastsq` instead of `fmin`. I don't recommend `leastsq` if `fmin` will suit your program equally well.

`numpy.polyfit`: Polynomial Fitting with NumPy

The function I used is `polyfit` from NumPy. The arguments it takes are

```
(x, y, deg, rcond=None, full=False, w=None, cov=False).
```

This function takes a `y` and an `x` signal and fits of polynomial of degree `deg` to it. In my project, I need fit a linear function to my data and get the sum of squared residual of this fitting. Originally, I planned use `scipy.optimize.curve_fit`, but this function doesn't return the residual as I desired. For `polyfit`, however, when we put the `full` argument to be `full=True`, the residual will be calculated for us. Together with it, the effective rank of the scaled Vandermonde coefficient matrix, its singular values. If desired, this function can also return the covariance matrix of the estimated polynomial coefficients. The diagonal elements of that matrix will be the variance of each coefficient.

–Zhang Zhang

HaarCascade: Feature Recognition in Images with SimpleCV

The HaarCascade package in SimpleCV is particularly useful. Specifically, I like the `findHaarFeatures` capability.

The `findHaarFeatures` function is called on an image and takes a training set as an argument. The function extracts and returns the desired feature from the image. The return also contains a Boolean value indicating whether the feature could be found in the image.

An example function call to look for a face in an image is below:

```
image = cam.getImage()
face = image.findHaarFeatures('face.xml')
if face:
      print 'There is a face in the image'
else:
      print 'There is not a face in the image'
```

I used the `findHaarFeatures` function in a very similar manner to the above. Namely, I used it to determine whether a face is present in an image taken by the webcam. Luckily, I didn't encounter any setbacks using the `findHaarFeatures` method.

–Jake Zeller

VPython: 3D Animation and 2D Real-time Graph Plotting

VPython allows 3D animation with a large number of vector objects (such as spheres, boxes, curves, etc.) at very fast speed. It also supports multiple display windows with real time updates on each of them. The windows can be either a 3D animation or a 2D graph-plotting. One cool thing is that the user can also change the perspective in the 3D animation window to change the perspective.

I used the 3D animation functionality to simulate the motion of particles interacting with user-defined potential in a box. At the same time, I used its graph-plotting functionality to get a real-update on the $g(r)$ against r plot for each of the simulations I ran. The multi-window support allows the user to run several simulations at the same time and compare the different features of interaction due to different temperature, particle size, particle number and the potential energy.

VPython is aesthetically appealing and relatively fast (3D rendering for 1000 particles!), has good 3D perspective change and multi-window support, and, most importantly, is easy to use.

One setback is that we can't run VPython from a normal terminal: instead we have to open a specific program and run its library from within. This also means I couldn't modulize the project; as such, I have to write the entire project in a very giant file.

–Alfred Zong