Histogramming and Binning Data with Python

Histogramming (see the file *Histogramming-example.py*)

When a measurement is made numerous times, it is often useful to bin (or group) the data and make a histogram. For example, if the time that it takes a sphere to roll down a ramp was measured one hundred times, then a histogram of the times would show how they are distributed.

The `hist` function from the `matplotlib` library is useful for making histograms. The first line of a program using it should be the following.

```
from pylab import *
```

Suppose `t` is an array of times. (there are 24 in the example program). The simplest way to make a histogram is to send the `hist` function only the data as shown below. As for other graphs, the `figure` and `show` functions are also needed. By default, the histogram will have 10 bins. If no additional arguments are sent, the `hist` function decides where to put the boundaries of the bins.

```
figure()
hist(t)
show()
```

The `hist` function returns the number of events in each bin, the edges of the bins, and things called patches (which will not be discussed further). These values can be captured by providing three variable names for them as follows.

```
events, edges, patches = hist(t)
```

For the example data, the `events` array is `[0, 0, 2, 2, 4, 13, 2, 1, 0, 0]`. The 10 elements are the numbers of events in the 10 bins. The array `edges` will contain 11 elements. For the example, the `edges` array is `[2.74, 3.206, 3.672, 4.138, 4.604, 5.07, 5.536, 6.002, 6.468, 6.934, 7.4]`. The first 10 elements are the lower edges of the bins and the final element is the upper edge of the final bin. The bins are the same width, but the edges may end up in unusual places.

You can control the number of bins by setting the `bins` argument to an integer as shown below, but this doesn't control the locations of the edges. Choosing an appropriate number of bins is important. If there are too few or too many bins, the histogram won't show how the events are distributed very well. Try histogramming the example data with 3 and 30 bins.

```
events, edges, patches = hist(t, bins=3)
```

If you want to have control over the number and location of the bins, you can make the `bins` argument an array. If you want N bins, the array will have $(N + 1)$ elements. The first N elements are the lower edges of the bins and the final element is the upper edge of the final bin. Usually the bins have equal widths, but they can be made unequal. The desired `bin` array can be made with the `linspace` function from the `scipy` library (you need the line "`from scipy import *`" near the beginning of the program). You must specify the first element of the array (the lower edge of the first bin), the last element of the array (the upper edge of the final bin), and the number of elements in the array (one more than the number of bins). The example below would produce 10 bins

(not 11) starting at 0 and ending at 10. For the example data, several of the bins will be empty.

```
bins = linspace(0, 10, 11)
events, edges, patches = hist(t, bins)
```

If you set the bins manually, it is a good idea to check that none of the data is being left out of the histogram. You can compare "`len(t)`", which is the length of the array `t` or the number of data points, and "`sum(events)`", which is the sum of all events included in the histogram.

It is also possible to set the upper and lower limits of the bins using the `range` argument. Values outside of the specified range are ignored. The following line should do the same as the previous example because the default number of bins is 10.

```
events, edges, patches = hist(t, range=(0.0,10.0) )
```

If you set the argument `normed` to "`True`", the function will make an area-normalized histogram. For each bin, the height on the histogram is the probability density, which is the number of events in the bin divided by the total number of events and the width of the bin. The area of each bin in the histogram is the probability of an event being in that bin, so the total area is one. With this option, the probability density is returned instead of the number of events.

```
probden, edges, patches = hist(t, bins, normed=True)
```

The `color` argument can be used to set the color of the bars in the histogram. Alternatively, the `edgecolor` and `facecolor` arguments separately set the colors of the edges and middle of the bars in the histogram, respectively. The `facecolor` argument can also be set to `none` so that the bars only have outlines. Some of other color options are:

`r` = red	`g` = green	`b` = blue	`k` = black
`c` = cyan	`m` = magenta	`y` = yellow	`w` = white

Binning Data (see the file *Binning-example.py*)

Sometimes data is binned before it is analyzed. For example, a set of decay times could be binned before fitting the data to an exponential function. The `histogram` function from the `numpy` library can be used to bin data without making a histogram. The first line of a program using it should be the following.

```
from numpy import *
```

The `histogram` function is similar to the `hist` function described in the previous section. The `bins` and `normed` arguments can be used, but it doesn't return `patches`. Associating the locations of the bins and the numbers of events in them is a little tricky because the `edges` array is one element longer than the `events` array. The example below will make an array called `tmid` which is the same length as `events` and contains the values of `t` in the middle of the bins. The `resize` function makes an array called `lower` which has a length one less than the length of the `edges` array. The new array contains the locations of the lower edges of the bins because the final element is dropped. An array containing the difference between consecutive elements of the `edges` array is

returned by "`diff(edges)`". Adding half of the difference between the edges to the lower edge gives the value in the middle of a bin. Note that `diff(edges)` is the same length as `lower`.

```
events, edges = histogram(t)
lower = resize(edges, len(edges)-1)
tmid =  lower + 0.5*diff(edges)
```

Additional documentation is available at:

http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.hist
http://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html