Reading and Writing Data Files with Python

In order plot or fit data with Python, you have to get the data to the program. If a program makes calculations using data, it can be useful to write the results to a file.

Reading Data Files:

Usually the data has to be in arrays. In the fitting examples, the data is entered directly into the programs. For instance, the following lines assign arrays of numbers to `x`, `y`, and `yerr`.

```
x = array([0.0, 2.0, 4.0, 6.0, 8.0])
y = array([1.1, 1.9, 3.2, 4.0, 5.9])
yerr = array([0.1, 0.2, 0.1, 0.3, 0.3])
```

This is not a very efficient way to handle large data sets. It would be better to store the data in a separate file from the program and have the program read the data file.

Use a text editor (*Idle* works well) to enter the data above in the following form. The values of `x`, `y`, and `yerr` (the uncertainty in `y`) for a single data point are entered on the same line separated by spaces or tabs.

```
0.0  1.1  0.1
2.0  1.9  0.2
4.0  3.2  0.1
6.0  4.0  0.3
8.0  5.9  0.3
```

Save the file as plain text and give it the name "`input.dat`". The `loadtxt` function from the `numpy` library is used to read data from a text file. The following lines read the data into an array called `DataIn`.

```
from numpy import *
DataIn = loadtxt('input.dat')
```

If you print `DataIn`, you will see that it is a 2-dimensional array. If you add the line below that starts with a number sign (#) to the data file, it will be ignored as a comment. Blank lines are also ignored. It is a good idea to put explanatory comments at the beginning of data files because you will quickly forget what the numbers mean. Giving files descriptive names is also helpful.

```
# This line is a comment, even in a data file
```

In most cases (plotting, for example), each variable should be in a separate 1-dimensional array. Setting the argument `unpack=True` and providing a variable for each column accomplishes this.

```
x, y, yerr = loadtxt('input.dat', unpack=True)
```

If you want to read in only some columns, you can use the `usecols` argument to specify which ones. Indices in Python start from zero, not one. The line below will read only the first and second columns of data, so only two variable names are provided.

```
x, y = loadtxt('input.dat',unpack=True, usecols=[0,1])
```

Writing Data Files:

The `savetxt` function from the `numpy` library is used to write data to a text file. Suppose that you've read two columns of data into the arrays `t` for time and `v` for the voltage from a pressure sensor. Also, suppose that the manual for the sensor gives the following equation to find the pressure in atmospheres from the voltage reading.

```
p = 0.15 + v/10.0
```

Recall that this single Python command will calculate an array `p` with the same length as the array `v`. Once you've calculated the pressures, you might want to write the times and pressures to a text file for later use. The following command will write `t` and `p` to the file "`output.dat`". The file will be saved in the same directory as the program**. If you give the name of an existing file, it will be overwritten so be careful!** Unfortunately, each of the arrays will appear in a different row, which is inconvenient for large data sets.

```
savetxt('output.dat', (t,p))
```

The `column_stack` function can be used to put each array written into a different column. The arguments should be a list of arrays (the inner pair of brackets make it a list) in the order that you want them to appear. The `column_stack` function stacks each of the arrays into a column of an array called `DataOut`, which is written to the text file.

```
DataOut = column_stack((t,p))
savetxt('output.dat', DataOut)
```

By default, the numbers will be written in scientific notation. The `fmt` argument can be used to specify the formatting. If one format is supplied, it will be used for all of the numbers. The general form of the `fmt` argument is

```
fmt = '%(width).(precision)(specifier)'
```

where `width` specifies the maximum number of digits, `precision` specifies the number of digits after the decimal point, and the possibilities for `specifier` are shown below. For integer formatting, the `precision` argument is ignored if you give it. For scientific notation and floating point formatting, the `width` argument is optional.

Specifier	Meaning	Example Format	Output for −34.5678
`i`	signed integer	`%5i`	`-34`
`e`	scientific notation	`%5.4e`	`-3.4568e+001`
`f`	floating point	`%5.2f`	`-34.57`

A format can also be provided for each column (two in this case) as follows.

```
savetxt('output.dat', DataOut, fmt=('%i3', '%4.3f'))
```

It is a good idea to add comments at the top of data files that you create to remind you what they contain. Unfortunately, the `savetxt` function doesn't provide do this. You could add descriptive comments using a text editor, but that is inconvenient.

A modified function called `savetxthd` is defined in the file "`newsavetxt.py`", which must be in the same directory as the example program. This function has the optional argument `header`, which allows you put comments at the top of the text file. Otherwise, it is the same as the `savetxt` function. To use the improved function, the following line must appear at the top of a program

```
from newsavetxt import *
```

An example is shown below. The string assigned to the `header` argument is written to the top of the output file. If you want the string to be considered a comment when it is read by the `loadtxt` function, it should start with a number sign (`#`). The `savetxthd` function doesn't automatically start a new line before outputting the data, so the text "`\n`" must appear in the string wherever you want a new line.

```
savetxthd('output.dat',DataOut,header='# A comment\n')
```

If you want a header with multiple lines, it is best to make the string beforehand as shown below. The "`+=`" operator appends the additional text to string `hdrtxt`. Each line should end with "`\n`" to force a newline.

```
hdrtxt ='# This is the first line\n'
hdrtxt += '# This is the second line\n'
savetxthd('output.dat', DataOut, header=hdrtxt)
```

Remember to be very careful about overwriting existing files with the `savetxt` (or `savetxthd`) function!

Additional documentation is available at:

http://docs.scipy.org/doc/numpy/reference/generated/numpy.loadtxt.html
http://docs.scipy.org/doc/numpy/reference/generated/numpy.savetxt.html
http://docs.scipy.org/doc/numpy/reference/generated/numpy.column_stack.html