Introduction to Numpy

Exercises and Solutions

Enthought, Inc.

CONTENTS

CHAPTER

ONE

EXERCISES

1.1 Numpy Exercises

1.1.1 Plotting

In PyLab, create a plot display that looks like the following:

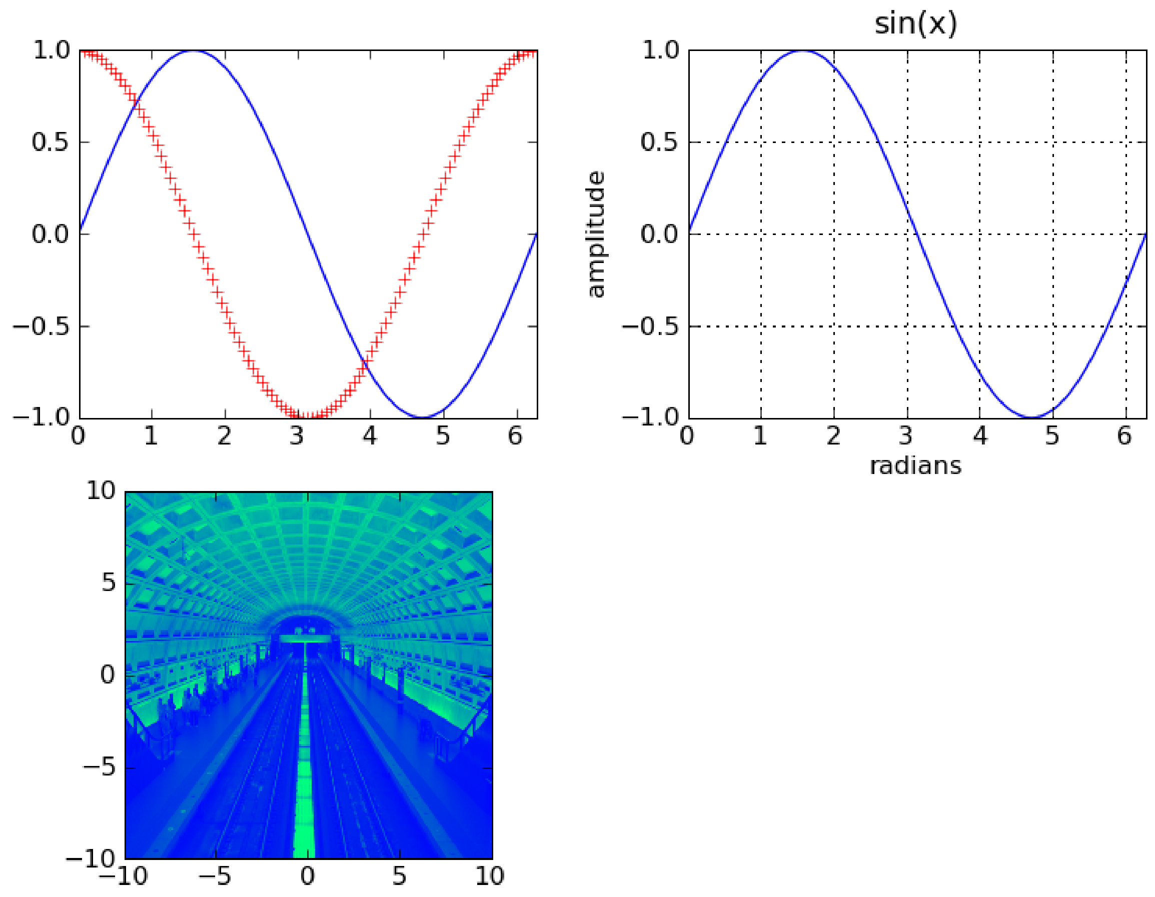

Photo credit: David Fettig

This is a 2x2 layout, with 3 slots occupied.

1. Sine function, with blue solid line; cosine with red '+' markers; the extents fit the plot exactly. Hint: see the axis() function for setting the extents.
2. Sine function, with gridlines, axis labels, and title; the extents fit the plot exactly.
3. Image with color map; the extents run from -10 to 10, rather than the default.

Save the resulting plot image to a file. (Use a different file name, so you don't overwrite the sample.)

The color map in the example is 'winter'; use 'cm.<tab>' to list the available ones, and experiment to find one you like.

Start with the following statements:

```
from scipy.misc.pilutil import imread

x = linspace(0, 2*pi, 101)
s = sin(x)
c = cos(x)

# 'flatten' creates a 2D array from a JPEG.
img = imread('dc_metro.jpg', flatten=True)
```

Tip: If you find that the label of one plot overlaps another plot, try adding a call to *tight_layout()* to your script.

Bonus

4. The *subplot()* function returns an axes object, which can be assigned to the *sharex* and *sharey* keyword arguments of another subplot() function call. E.g.:

   ```
   ax1 = subplot(2,2,1)
   ...
   subplot(2,2,2, sharex=ax1, sharey=ax1)
   ```

   Make this modification to your script, and explore the consequences. Hint: try panning and zooming in the subplots.

See *Plotting - Solution 1*.

1.1.2 Calculate Derivative

Topics: NumPy array indexing and array math.

Use array slicing and math operations to calculate the numerical derivative of `sin` from 0 to `2*pi`. There is no need to use a 'for' loop for this.

Plot the resulting values and compare to `cos`.

Bonus

Implement integration of the same function using Riemann sums or the trapezoidal rule.

See *Calculate Derivative - Solution*.

1.1.3 Load Array from Text File

0. From the IPython prompt, type:

   ```
   In [1]: loadtxt?
   ```

   to see the options on how to use the loadtxt command.

1. Use loadtxt to load in a 2D array of floating point values from 'float_data.txt'. The data in the file looks like:

   ```
   1 2 3 4
   5 6 7 8
   ```

   The resulting data should be a 2x4 array of floating point values.

2. In the second example, the file 'float_data_with_header.txt' has strings as column names in the first row:

   ```
   c1 c2 c3 c4
    1  2  3  4
    5  6  7  8
   ```

   Ignore these column names, and read the remainder of the data into a 2D array.

   Later on, we'll learn how to create a "structured array" using these column names to create fields within an array.

Bonus

3. A third example is more involved. It contains comments in multiple locations, uses multiple formats, and includes a useless column to skip:

   ```
   -- THIS IS THE BEGINNING OF THE FILE --
   % This is a more complex file to read!

   % Day,  Month,  Year, Useless Col, Avg Power
      01,     01,  2000,      ad766,         30
      02,     01,  2000,       t873,         41
   % we don't have Jan 03rd!
      04,     01,  2000,       r441,         55
      05,     01,  2000,       s345,         78
      06,     01,  2000,       x273,        134 % that day was crazy
      07,     01,  2000,       x355,         42

   %-- THIS IS THE END OF THE FILE --
   ```

See *Load Array from Text File - Solution*

1.1.4 Filter Image

Read in the "dc_metro" image and use an averaging filter to "smooth" the image. Use a "5 point stencil" where you average the current pixel with its neighboring pixels:

```
0 0 0 0 0 0 0
0 0 0 x 0 0 0
0 0 x x x 0 0
0 0 0 x 0 0 0
0 0 0 0 0 0 0
```

Plot the image, the smoothed image, and the difference between the two.

Bonus

Re-filter the image by passing the result image through the filter again. Do this 50 times and plot the resulting image.

See *Filter Image - Solution*.

1.1.5 Dow Selection

Topics: Boolean array operators, sum function, where function, plotting.

The array 'dow' is a 2-D array with each row holding the daily performance of the Dow Jones Industrial Average from the beginning of 2008 (dates have been removed for exercise simplicity). The array has the following structure:

```
OPEN      HIGH      LOW       CLOSE     VOLUME      ADJ_CLOSE
13261.82  13338.23  12969.42  13043.96  3452650000  13043.96
13044.12  13197.43  12968.44  13056.72  3429500000  13056.72
13046.56  13049.65  12740.51  12800.18  4166000000  12800.18
12801.15  12984.95  12640.44  12827.49  4221260000  12827.49
12820.9   12998.11  12511.03  12589.07  4705390000  12589.07
12590.21  12814.97  12431.53  12735.31  5351030000  12735.31
```

0. The data has been loaded from a .csv file for you.
1. Create a "mask" array that indicates which rows have a volume greater than 5.5 billion.
2. How many are there? (hint: use sum).
3. Find the index of every row (or day) where the volume is greater than 5.5 billion. hint: look at the where() command.

Bonus

1. Plot the adjusted close for *every* day in 2008.
2. Now over-plot this plot with a 'red dot' marker for every day where the dow was greater than 5.5 billion.

See *Dow Selection - Solution*.

1.1.6 Wind Statistics

Topics: Using array methods over different axes, fancy indexing.

1. The data in 'wind.data' has the following format:

   ```
   61  1  1 15.04 14.96 13.17  9.29 13.96  9.87 13.67 10.25 10.83 12.58 18.50 15.04
   61  1  2 14.71 16.88 10.83  6.50 12.62  7.67 11.50 10.04  9.79  9.67 17.54 13.83
   61  1  3 18.50 16.88 12.33 10.13 11.17  6.17 11.25  8.04  8.50  7.67 12.75 12.71
   ```

   The first three columns are year, month and day. The remaining 12 columns are average windspeeds in knots at 12 locations in Ireland on that day.

   Use the 'loadtxt' function from numpy to read the data into an array.
2. Calculate the min, max and mean windspeeds and standard deviation of the windspeeds over all the locations and all the times (a single set of numbers for the entire dataset).
3. Calculate the min, max and mean windspeeds and standard deviations of the windspeeds at each location over all the days (a different set of numbers for each location)

4. Calculate the min, max and mean windspeed and standard deviations of the windspeeds across all the locations at each day (a different set of numbers for each day)
5. Find the location which has the greatest windspeed on each day (an integer column number for each day).
6. Find the year, month and day on which the greatest windspeed was recorded.
7. Find the average windspeed in January for each location.

You should be able to perform all of these operations without using a for loop or other looping construct.

Bonus

1. Calculate the mean windspeed for each month in the dataset. Treat January 1961 and January 1962 as *different* months. (hint: first find a way to create an identifier unique for each month. The second step might require a for loop.)
2. Calculate the min, max and mean windspeeds and standard deviations of the windspeeds across all locations for each week (assume that the first week starts on January 1 1961) for the first 52 weeks. This can be done without any for loop.

Bonus Bonus

Calculate the mean windspeed for each month without using a for loop. (Hint: look at *searchsorted* and *add.reduceat*.)

Notes

These data were analyzed in detail in the following article:

> Haslett, J. and Raftery, A. E. (1989). Space-time Modelling with Long-memory Dependence: Assessing Ireland's Wind Power Resource (with Discussion). Applied Statistics 38, 1-50.

See *Wind Statistics - Solution*.

1.1.7 Sinc Function

Topics: Broadcasting, Fancy Indexing

Calculate the sinc function: sin(r)/r. Use a Cartesian x,y grid and calculate `r = sqrt(x**2+y**2)` with 0 in the center of the grid. Calculate the function for -15,15 for both x and y.

See *Sinc Function - Solution*.

1.1.8 Structured Array

Topics: Read columns of data into a structured array using loadtxt.

1. The data in 'short_logs.crv' has the following format:

```
DEPTH          CALI        S-SONIC    ...
8744.5000   -999.2500   -999.2500    ...
8745.0000   -999.2500   -999.2500    ...
8745.5000   -999.2500   -999.2500    ...
```

Here the first row defines a set of names for the columns of data in the file. Use these column names to define a dtype for a structured array that will have fields 'DEPTH', 'CALI', etc. Assume all the data is of the float64 data format.

2. Use the 'loadtxt' method from numpy to read the data from the file into a structured array with the dtype created in (1). Name this array 'logs'
3. The 'logs' array is nice for retrieving columns from the data. For example, logs['DEPTH'] returns the values from the DEPTH column of the data. For row-based or array-wide operations, it is more convenient to have a 2D view into the data, as if it is a simple 2D array of float64 values.

   Create a 2D array called 'logs_2d' using the view operation. Be sure the 2D array has the same number of columns as in the data file.
4. -999.25 is a "special" value in this data set. It is intended to represent missing data. Replace all of these values with NaNs. Is this easier with the 'logs' array or the 'logs_2d' array?
5. Create a mask for all the "complete" rows in the array. A complete row is one that doesn't have any NaN values measured in that row.

   HINT: The `all` function is also useful here.
6. Plot the VP vs VS logs for the "complete" rows.

See *Structured Array - Solution*.

CHAPTER

TWO

SOLUTIONS

2.1 Numpy - Solutions

2.1.1 Plotting - Solution 1

plotting_bonus_solution.py:

```
"""
Plotting
--------

In PyLab, create a plot display that looks like the following:

.. image:: plotting/sample_plots.png

`Photo credit: David Fettig <http://www.publicdomainpictures.net/view-image.php?image=507>`_


This is a 2x2 layout, with 3 slots occupied.

1. Sine function, with blue solid line; cosine with red '+' markers; the extents
   fit the plot exactly. Hint: see the axis() function for setting the extents
2. Sine function, with gridlines, axis labels, and title; the extents fit the
   plot exactly.
3. Image with color map; the extents run from -10 to 10, rather than the
   default.

Save the resulting plot image to a file. (Use a different file name, so you
don't overwrite the sample.)

The color map in the example is 'winter'; use 'cm.<tab>' to list the available
ones, and experiment to find one you like.

Start with the following statements::

    from scipy.misc.pilutil import imread

    x = linspace(0, 2*pi, 101)
    s = sin(x)
    c = cos(x)

    # 'flatten' creates a 2D array from a JPEG.
    img = imread('dc_metro.jpg', flatten=True)
```

```
Tip: If you find that the label of one plot overlaps another plot, try adding
a call to `tight_layout()` to your script.

Bonus
~~~~~

4. The `subplot()` function returns an axes object, which can be assigned to
   the `sharex` and `sharey` keyword arguments of another subplot() function
   call.  E.g.::

       ax1 = subplot(2,2,1)
       ...
       subplot(2,2,2, sharex=ax1, sharey=ax1)

   Make this modification to your script, and explore the consequences.
   Hint: try panning and zooming in the subplots.

"""


# The following imports are *not* needed in PyLab, but are needed in this file.
from numpy import linspace, pi, sin, cos
from pylab import plot, subplot, cm, imshow, xlabel, ylabel, title, grid, \
                  axis, show, savefig, gcf, figure, close, tight_layout

# The following import *is* needed in PyLab.
# The PyLab version of 'imread' does not read JPEGs.
from scipy.misc.pilutil import imread

x = linspace(0, 2*pi, 101)
s = sin(x)
c = cos(x)

# 'flatten' creates a 2D array from a JPEG.
img = imread('dc_metro.JPG', flatten=True)

close('all')
# 2x2 layout, first plot: sin and cos
ax1 = subplot(2,2,1)
plot(x, s, 'b-', x, c, 'r+')
axis('tight')

# 2nd plot: gridlines, labels
subplot(2,2,2, sharex=ax1, sharey=ax1)
plot(x, s)
grid()
xlabel('radians')
ylabel('amplitude')
title('sin(x)')
axis('tight')

# 3rd plot, image
subplot(2,2,3)
imshow(img, extent=[-10, 10, -10, 10], cmap=cm.winter)

tight_layout()

show()
```

```
savefig('my_plots.png')
```

2.1.2 Calculate Derivative - Solution

calc_derivative_solution.py:

```
"""
Topics: NumPy array indexing and array math.

Use array slicing and math operations to calculate the
numerical derivative of ``sin`` from 0 to ``2*pi``.  There is no
need to use a for loop for this.

Plot the resulting values and compare to ``cos``.

Bonus
~~~~~

Implement integration of the same function using Riemann sums or the
trapezoidal rule.

"""
from numpy import linspace, pi, sin, cos, cumsum
from pylab import plot, show, subplot, legend, title

# calculate the sin() function on evenly spaced data.
x = linspace(0,2*pi,101)
y = sin(x)

# calculate the derivative dy/dx numerically.
# First, calculate the distance between adjacent pairs of
# x and y values.
dy = y[1:]-y[:-1]
dx = x[1:]-x[:-1]

# Now divide to get "rise" over "run" for each interval.
dy_dx = dy/dx

# Assuming central differences, these derivative values
# centered in-between our original sample points.
centers_x = (x[1:]+x[:-1])/2.0

# Plot our derivative calculation.  It should match up
# with the cos function since the derivative of sin is
# cos.
subplot(1,2,1)
plot(centers_x, dy_dx,'rx', centers_x, cos(centers_x),'b-')
title(r"$\rm{Derivative\ of}\ sin(x)$")

# Trapezoidal rule integration.
avg_height = (y[1:]+y[:-1])/2.0
int_sin = cumsum(dx * avg_height)

# Plot our integration against -cos(x) - -cos(0)
closed_form = -cos(x)+cos(0)
```

```
subplot(1,2,2)
plot(x[1:], int_sin,'rx', x, closed_form,'b-')
legend(('numerical', 'actual'))
title(r"$\int \, \sin(x) \, dx$")
show()
```

2.1.3 Load Array from Text File - Solution

load_text_solution.py:

```
"""
Load Array from Text File
-------------------------

0. From the IPython prompt, type::

        In [1]: loadtxt?

   to see the options on how to use the loadtxt command.


1. Use loadtxt to load in a 2D array of floating point values from
   'float_data.txt'.  The data in the file looks like::

        1 2 3 4
        5 6 7 8

   The resulting data should be a 2x4 array of floating point values.

2. In the second example, the file 'float_data_with_header.txt' has
   strings as column names in the first row::

        c1 c2 c3 c4
         1  2  3  4
         5  6  7  8

   Ignore these column names, and read the remainder of the data into
   a 2D array.

   Later on, we'll learn how to create a "structured array" using
   these column names to create fields within an array.

Bonus
~~~~~

3. A third example is more involved. It contains comments in multiple
   locations, uses multiple formats, and includes a useless column to
   skip::

    -- THIS IS THE BEGINNING OF THE FILE --
    % This is a more complex file to read!

    % Day,  Month,  Year, Useless Col, Avg Power
       01,     01,  2000,      ad766,         30
       02,     01,  2000,       t873,         41
    % we don't have Jan 03rd!
       04,     01,  2000,       r441,         55
```

```
       05,     01,  2000,       s345,         78
       06,     01,  2000,       x273,        134 % that day was crazy
       07,     01,  2000,       x355,         42

    %-- THIS IS THE END OF THE FILE --
"""

from numpy import loadtxt

#############################################################################
# 1. Simple example loading a 2x4 array of floats from a file.
#############################################################################
ary1 = loadtxt('float_data.txt')

print 'example 1:'
print ary1


#############################################################################
# 2. Same example, but skipping the first row of column headers
#############################################################################
ary2 = loadtxt('float_data_with_header.txt', skiprows=1)

print 'example 2:'
print ary2

#############################################################################
# 3. More complex example with comments and columns to skip
#############################################################################
ary3 = loadtxt("complex_data_file.txt", delimiter=",", comments="%",
               usecols=(0,1,2,4), dtype=int, skiprows=1)

print 'example 3:'
print ary3
```

2.1.4 Filter Image - Solution

filter_image_solution.py:

```
"""
Filter Image
------------

Read in the "dc_metro" image and use an averaging filter
to "smooth" the image.  Use a "5 point stencil" where
you average the current pixel with its neighboring pixels::

              0 0 0 0 0 0 0
              0 0 0 x 0 0 0
              0 0 x x x 0 0
              0 0 0 x 0 0 0
              0 0 0 0 0 0 0

Plot the image, the smoothed image, and the difference between the
two.

Bonus
```

```
~~~~~

Re-filter the image by passing the result image through the filter again. Do
this 50 times and plot the resulting image.

"""

from scipy.misc.pilutil import imread
from pylab import figure, subplot, imshow, title, show, gray, cm

def smooth(img):
    avg_img =(  img[1:-1 ,1:-1]  # center
              + img[ :-2 ,1:-1]  # top
              + img[2:   ,1:-1]  # bottom
              + img[1:-1 , :-2]  # left
              + img[1:-1 ,2:  ]  # right
              ) / 5.0
    return avg_img

# 'flatten' creates a 2D array from a JPEG.
img = imread('dc_metro.JPG', flatten=True)
avg_img = smooth(img)


figure()
# Set colormap so that images are plotted in gray scale.
gray()
# Plot the original image first
subplot(1,3,1)
imshow(img)
title('original')

# Now the filtered image.
subplot(1,3,2)
imshow(avg_img)
title('smoothed once')

# And finally the difference between the two.
subplot(1,3,3)
imshow(img[1:-1,1:-1] - avg_img)
title('difference')


# Bonus: Re-filter the image by passing the result image
#        through the filter again.  Do this 50 times and plot
#        the resulting image.

for num in range(50):
    avg_img = smooth(avg_img)

print avg_img.shape, img.shape

# Plot the original image first
figure()
subplot(1,2,1)
imshow(img)
title('original')
```

```
# Now the filtered image.
subplot(1,2,2)
imshow(avg_img)
title('smoothed 50 times')

show()
```

2.1.5 Dow Selection - Solution

dow_selection_solution.py:

```
"""

Topics: Boolean array operators, sum function, where function, plotting.

The array 'dow' is a 2-D array with each row holding the
daily performance of the Dow Jones Industrial Average from the
beginning of 2008 (dates have been removed for exercise simplicity).
The array has the following structure::

       OPEN      HIGH      LOW       CLOSE     VOLUME      ADJ_CLOSE
       13261.82  13338.23  12969.42  13043.96  3452650000  13043.96
       13044.12  13197.43  12968.44  13056.72  3429500000  13056.72
       13046.56  13049.65  12740.51  12800.18  4166000000  12800.18
       12801.15  12984.95  12640.44  12827.49  4221260000  12827.49
       12820.9   12998.11  12511.03  12589.07  4705390000  12589.07
       12590.21  12814.97  12431.53  12735.31  5351030000  12735.31

0. The data has been loaded from a .csv file for you.
1. Create a "mask" array that indicates which rows have a volume
   greater than 5.5 billion.
2. How many are there?  (hint: use sum).
3. Find the index of every row (or day) where the volume is greater
   than 5.5 billion. hint: look at the where() command.

Bonus
~~~~~

1. Plot the adjusted close for *every* day in 2008.
2. Now over-plot this plot with a 'red dot' marker for every
   day where the dow was greater than 5.5 billion.

"""

from numpy import where, loadtxt
from pylab import figure, hold, plot, show

# Constants that indicate what data is held in each column of
# the 'dow' array.
OPEN  = 0
HIGH = 1
LOW = 2
CLOSE = 3
VOLUME = 4
ADJ_CLOSE = 5

# 0. The data has been loaded from a csv file for you.
```

```
# 'dow' is our NumPy array that we will manipulate.
dow = loadtxt('dow.csv', delimiter=',')


# 1. Create a "mask" array that indicates which rows have a volume
#    greater than 5.5 billion.
high_volume_mask = dow[:,VOLUME] > 5.5e9

# 2. How many are there?  (hint: use sum).
high_volume_days = sum(high_volume_mask)
print "The dow volume has been above 5.5 billion on" \
      " %d days this year." % high_volume_days

# 3. Find the index of every row (or day) where the volume is greater
#    than 5.5 billion. hint: look at the where() command.
high_vol_index = where(high_volume_mask)[0]

# BONUS:
# 1. Plot the adjusted close for EVERY day in 2008.
# 2. Now over-plot this plot with a 'red dot' marker for every
#    day where the dow was greater than 5.5 billion.

# Create a new plot.
figure()

# Plot the adjusted close for every day of the year as a blue line.
# In the format string 'b-', 'b' means blue and '-' indicates a line.
plot(dow[:,ADJ_CLOSE],'b-')

# Plot the days where the volume was high with red dots...
plot(high_vol_index, dow[high_vol_index, ADJ_CLOSE],'ro')

# Scripts must call the plot "show" command to display the plot
# to the screen.
show()
```

2.1.6 Wind Statistics - Solution

wind_statistics_solution.py:

```
"""
Wind Statistics
----------------

Topics: Using array methods over different axes, fancy indexing.

1. The data in 'wind.data' has the following format::

        61  1  1 15.04 14.96 13.17  9.29 13.96  9.87 13.67 10.25 10.83 12.58 18.50 15.04
        61  1  2 14.71 16.88 10.83  6.50 12.62  7.67 11.50 10.04  9.79  9.67 17.54 13.83
        61  1  3 18.50 16.88 12.33 10.13 11.17  6.17 11.25  8.04  8.50  7.67 12.75 12.71

   The first three columns are year, month and day.  The
   remaining 12 columns are average windspeeds in knots at 12
   locations in Ireland on that day.
   Use the 'loadtxt' function from numpy to read the data into
   an array.

2. Calculate the min, max and mean windspeeds and standard deviation of the
   windspeeds over all the locations and all the times (a single set of numbers
   for the entire dataset).

3. Calculate the min, max and mean windspeeds and standard deviations of the
   windspeeds at each location over all the days (a different set of numbers
   for each location)

4. Calculate the min, max and mean windspeed and standard deviations of the windspeeds
   across all the locations at each day (a different set of numbers for each day)

5. Find the location which has the greatest windspeed on each day (an integer column
   number for each day).

6. Find the year, month and day on which the greatest windspeed was recorded.

7. Find the average windspeed in January for each location.

You should be able to perform all of these operations without using a for
loop or other looping construct.

Bonus
~~~~~

1. Calculate the mean windspeed for each month in the dataset.  Treat
   January 1961 and January 1962 as *different* months.

2. Calculate the min, max and mean windspeeds and standard deviations of the
   windspeeds across all locations for each week (assume that the first week
   starts on January 1 1961) for the first 52 weeks.

Bonus Bonus
~~~~~~~~~~~

Calculate the mean windspeed for each month without using a for loop.
(Hint: look at `searchsorted` and `add.reduceat`.)

Notes
~~~~~

These data were analyzed in detail in the following article:

   Haslett, J. and Raftery, A. E. (1989). Space-time Modelling with
   Long-memory Dependence: Assessing Ireland's Wind Power Resource
   (with Discussion). Applied Statistics 38, 1-50.

"""

from numpy import loadtxt, arange, searchsorted, add, zeros

wind_data = loadtxt('wind.data')

data = wind_data[:,3:]

print '2. Statistics over all values'
```

```
print '  min:', data.min()
print '  max:', data.max()
print '  mean:', data.mean()
print '  standard deviation:', data.std()
print

print '3. Statistics over all days at each location'
print '  min:', data.min(axis=0)
print '  max:', data.max(axis=0)
print '  mean:', data.mean(axis=0)
print '  standard deviation:', data.std(axis=0)
print

print '4. Statistics over all locations for each day'
print '  min:', data.min(axis=1)
print '  max:', data.max(axis=1)
print '  mean:', data.mean(axis=1)
print '  standard deviation:', data.std(axis=1)
print

print '5. Statistics over all days at each location'
print '  daily max location:', data.argmax(axis=1)
print

daily_max = data.max(axis=1)
max_row = daily_max.argmax()

print '6. Day of maximum reading'
print '  Year:', int(wind_data[max_row,0])
print '  Month:', int(wind_data[max_row,1])
print '  Day:', int(wind_data[max_row,2])
print

january_indices = wind_data[:,1] == 1
january_data = data[january_indices]

print '7. Statistics for January'
print '  mean:', january_data.mean(axis=0)
print

# Bonus

# compute the month number for each day in the dataset
months = (wind_data[:,0]-61)*12 + wind_data[:,1] - 1

# get set of unique months
month_values = set(months)

# initialize an array to hold the result
monthly_means = zeros(len(month_values))

for month in month_values:
    # find the rows that correspond to the current month
    day_indices = (months == month)

    # extract the data for the current month using fancy indexing
    month_data = data[day_indices]
    # find the mean
    monthly_means[month] = month_data.mean()

    # Note: experts might do this all-in one
    # monthly_means[month] = data[months==month].mean()

# In fact the whole for loop could reduce to the following one-liner
# monthly_means = array([data[months==month].mean() for month in month_values])


print "Bonus 1."
print "  mean:", monthly_means
print

# Bonus 2.
# Extract the data for the first 52 weeks. Then reshape the array to put
# on the same line 7 days worth of data for all locations. Let Numpy
# figure out the number of lines needed to do so
weekly_data = data[:52*7].reshape(-1, 7*12)

print 'Bonus 2. Weekly statistics over all locations'
print '  min:', weekly_data.min(axis=1)
print '  max:', weekly_data.max(axis=1)
print '  mean:', weekly_data.mean(axis=1)
print '  standard deviation:', weekly_data.std(axis=1)
print

# Bonus Bonus : this is really tricky...

# compute the month number for each day in the dataset
months = (wind_data[:,0]-61)*12 + wind_data[:,1] - 1

# find the indices for the start of each month
# this is a useful trick - we use range from 0 to the
# number of months + 1 and searchsorted to find the insertion
# points for each.
month_indices = searchsorted(months, arange(months[-1]+2))

# now use add.reduceat to get the sum at each location
monthly_loc_totals = add.reduceat(data, month_indices[:-1])

# now use add to find the sum across all locations for each month
monthly_totals = monthly_loc_totals.sum(axis=1)

# now find total number of measurements for each month
month_days = month_indices[1:] - month_indices[:-1]
measurement_count = month_days*12

# compute the mean
monthly_means = monthly_totals/measurement_count

print "Bonus Bonus"
print "  mean:", monthly_means

# Notes: this method relies on the fact that the months are contiguous in the
# data set - the method used in the bonus section works for non-contiguous
# days.
```

2.1.7 Sinc Function - Solution

sinc_function_solution.py:

```
"""
Topics: Broadcasting, Fancy Indexing

Calculate the sinc function: sin(r)/r.  Use a Cartesian x,y grid
and calculate ``r = sqrt(x**2+y**2)`` with 0 in the center of the grid.
Calculate the function for -15,15 for both x and y.
"""

from numpy import linspace, sin, sqrt, newaxis
from pylab import imshow, gray, show

x = linspace(-15,15,101)
# flip y up so that it is a "column" vector.
y = linspace(-15,15,101)[:,newaxis]

# because of broadcasting rules, r is 2D.
r = sqrt(x**2+y**2)

# calculate our function.
sinc = sin(r)/r

# replace any location where r is 0 with 1.0
sinc[r==0] = 1.0

imshow(sinc, extent=[-15,15,-15,15])
gray()
show()
```

2.1.8 Structured Array - Solution

structured_array_solution.py:

```
""" Read columns of data into a structured array using loadtxt.

    1. The data in 'short_logs.crv' has the following format::

           DEPTH         CALI       S-SONIC   ...
           8744.5000   -999.2500   -999.2500   ...
           8745.0000   -999.2500   -999.2500   ...
           8745.5000   -999.2500   -999.2500   ...

       Here the first row defines a set of names for the columns
       of data in the file.  Use these column names to define a
       dtype for a structured array that will have fields 'DEPTH',
       'CALI', etc.  Assume all the data is of the float64 data
       format.

    2. Use the 'loadtxt' method from numpy to read the data from
       the file into a structured array with the dtype created
       in (1).  Name this array 'logs'

    3. The 'logs' array is nice for retrieving columns from the data.
       For example, logs['DEPTH'] returns the values from the DEPTH
```

```
        column of the data.  For row-based or array-wide operations,
        it is more convenient to have a 2D view into the data, as if it
        is a simple 2D array of float64 values.

        Create a 2D array called 'logs_2d' using the view operation.
        Be sure the 2D array has the same number of columns as in the
        data file.

     4. -999.25 is a "special" value in this data set.  It is
        intended to represent missing data.  Replace all of these
        values with NaNs.  Is this easier with the 'logs' array
        or the 'logs_2d' array?

     5. Create a mask for all the "complete" rows in the array.
        A complete row is one that doesn't have any NaN values measured
        in that row.

        HINT: The ``all`` function is also useful here.

     6. Plot the VP vs VS logs for the "complete" rows.
"""
from numpy import dtype, loadtxt, float64, NaN, isfinite, all
from pylab import plot, show, xlabel, ylabel

# Open the file.
log_file = open('short_logs.crv')

# 1.Create a dtype from the names in the file header.
header = log_file.readline()
log_names = header.split()

# Construct the array "dtype" that describes the data.  All fields
# are 8 byte (64 bit) floating point.
fields = zip(log_names, ['f8']*len(log_names))
fields_dtype = dtype(fields)

#2. Use loadtxt to load the data into a structured array.
logs = loadtxt(log_file, dtype=fields_dtype)

# 3. Make a 2D, float64 view of the data.
#    The -1 value for the row shape means that numpy should
#    make this dimension whatever it needs to be so that
#    rows*cols = size for the array.
values = logs.view(float64)
values.shape = -1, len(fields)

# 4. Relace any values that are -999.25 with NaNs.
values[values==-999.25] = NaN

# 5. Make a mask for all the rows that don't have any missing values.
#    Pull out these samples from the logs array into a separate array.
data_mask = all(isfinite(values), axis=-1)
good_logs = logs[data_mask]


# 6. Plot VP vs. VS for the "complete rows.
plot(good_logs['VS'], good_logs['VP'], 'o')
xlabel('VS')
```

```
ylabel('VP')
show()
```

This document was generated on July 10, 2012.