PHYS2300: Assignment 4 - Interpolation

Objective

The purpose of this tutorial is to exercise your ability to read, correlate, and interpolate data sets. By the end of this tutorial, you will be able to integrate two different datasets, one of temperature vs. time and one of altitude vs. time, to create an interpolated dataset of temperatures vs. altitude for a high altitude balloon flight.

Prerequisites: Assignments 1, 2 and 3

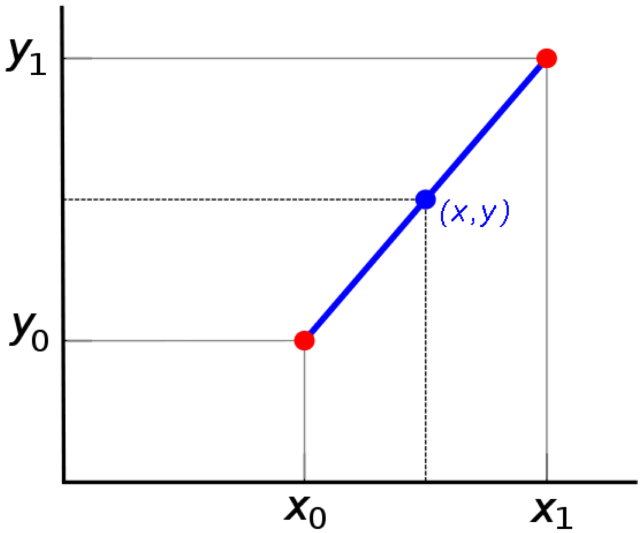
Introduction

Experiments produce data, and sometimes the interpretation of that data depends on other measurements from other experiments. For example, the High Altitude Ballooning Team, HARBOR flies a helium balloon to over 100,000 ft taking measurements of Earth's atmosphere along the way. At the same time, a GPS receiver onboard charts the balloon payload's location in real time and broadcasts that to a receiving computer on the ground so that the launch team can chase, locate, and recover the payload after flight. Each measurement records what scientists call a time series, that is a table of data and the associated time that measurement was recorded. A temperature sensor on the payload records the temperature over time - say every few seconds - and the GPS receiver records the balloon's altitude over time. If we can combine these two data sets, we'd have an interesting measurement of the temperature at each altitude during the balloon's flight.
But there is a problem. The temperature measurements are regular, but because the GPS receiver is broadcasting the altitude to the ground, it is not only less frequently measured, but the timing isn't regular due to broadcasting issues with the transmitter involved. So we can't just simply match up the data from one and the data from another and make a nice plot of temperature vs. altitude. Instead, we must use the time of the temperature measurement to interpolate the altitude from the more sparsely sampled GPS data.

A note about interpolation

Interpolation is the process we use to determine an estimate of the value of a measured quantity between two known values. For example, in the figure below, we have made two measurements, y_0, y_1 at two different values of x_0, x_1.

We can determine an estimate for any value of y at any value of x anywhere between x_0,x_1 by finding the slope and intercept of the line connecting the two points and using that to compute the value we are interested in. This is known as linear interpolation, and, for most well sampled, smoothly varying measurements, like our altitude data, this is an excellent way to approximate the interpolated value. And it is the one we will use in this



tutorial. Recalling the equation for a line, y = mx + b, take a moment to work out expressions for m and b in terms of y_0,y_1 and x_0,x_1. You'll need those later.

Designing your code

1. Develop our problem statement: "We need a code that will read in temperature time series, read in a GPS time series, and use an interpolation algorithm to produce a plot of temperature vs. altitude."
2. Define the inputs and outputs: We will use the following input data files (see the Canvas assignment page):
    - Atmospheric data - Data from the HARBOR PASCAL instrument that logs the temperature as a function of time.
    - GPS data - The transmission log file from the inflight telemetry data

    Go ahead and download these. The output will be a plot of temperature vs. altitude, one panel for the ascent stage of the flight, and the other for the descent stage. Note the first file contains three columns of temperature data. We will be using the first temperature column, as this was the sensor taped to the outside of the payload.
3. Develop the algorithm: The algorithm will build on your skills manipulating files from the last tutorial, and add in the interpolation scheme above. As we read in each temperature measurement, we'll need to compare that to the altitude data, located the GPS times that bound the temperature data, and perform the interpolation. The interpolated altitude will be stored as a pair with that temperature to create a dataset we can plot. We'll refer to this as the "correlated" data, as compared to the "raw" data. We do this because any interpolation is merely an estimate of the data, not an actual measurement itself.
4. Translate the algorithm into pseudocode and, ultimately, python: By now, you are getting a pretty good hang of how to get these things going: you write a completely working python shell that does nothing but list the things you need to do. However, I am going to go a bit further this time, and instead of just writing a comment line for what I want the code to do, I will specify a function that will perform that action. So, looking at the following:

```
#-----------------------------------------------------------------------
# G e n e r a l   I n f o r m a t i o n
#-----------------------------------------------------------------------
# Name: harbor.py
#
# Usage: python harbor.py
#
# Description: Code to correlate and interpolate temperature vs. time with
altitude vs. time
#
# Inputs: Wxfile: contains the HARBOR weather data (WX)
#         GPSfile: contains the HARBOR GPS track data
#         DisplayOption: either "show" to screen or "save" to file
#
# Outputs: plots and analysis
#
# Auxiliary Files: None
#
# Special Instructions: None
#
#-----------------------------------------------------------------------
```

```
# C o d e   H i s t o r y
#-------------------------------------------------------------------------------
# Version: 1.0
#
# Author(s): John Armstrong (jca)
#            jcarmstrong@weber.edu
#
# Modifications:
# 5 JUL 2012 - First Light (jca)
#
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# I m p o r t   L i b r a r i e s
#-------------------------------------------------------------------------------

import sys
import matplotlib
import matplotlib.pylab as plot
import math as m
import numpy as np

#-------------------------------------------------------------------------------
# D e f i n e   f u n c t i o n s
#-------------------------------------------------------------------------------

def readWxData(wxFileName):
        # Do Stuff
def readGPSData(gpsFileName):
        # Do Stuff
def interpolateWxFromGPS(wxTimes,gpsTimes,gpsAltitudes):
        # Do Stuff
def plotAllFigs(display):
        # Do Stuff


#-------------------------------------------------------------------------------
# M a i n    P r o g r a m
#-------------------------------------------------------------------------------

# parse the name of the data file

try:
    wxFileName = 'TempAndPressure.tx'
    gpsFileName ='gpsData.txt'
    display = 'show' # or save
except (ValueError, IndexError), e:
    print e
    sys.exit()

# read in temperature and time data

wxTimes, wxTemperatures = readWxData(wxFileName)
gpsTimes, gpsAltitudes = readGPSData(gpsFileName)

# compute wx alts by interpolating from gps alts
```

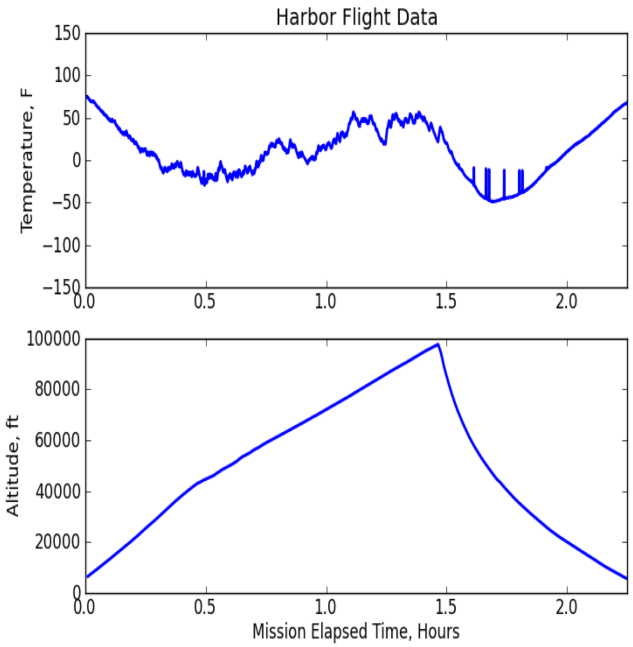
```
wxCorrelatedAltitudesUp, wxCorrelatedTemperaturesUp,
wxCorrelatedAltitudesDown, wxCorrelatedTemperaturesDown = \
                          interpolateWxFromGPS(wxTimes,gpsTimes,gpsAltitudes)
# plot and quit

plotAllFigs(display)
```

This is a perfectly functional python code that does absolutely nothing useful, except it does give us a good guide for how to proceed:

- After my standard header, I write down the shells of a few functions I think I'll need, along with very descriptive names, so their name tells me exactly what I want them to do.
- Next, in the main program, I have a block for reading in the names of the two files plus a keyword I call display. I want to use this to tell the program whether to *show* the graph in the screen or *save* the graph to a file.
- Finally, I write the rest of the program in terms of the functions that will perform the duties. This gives me a clean, modular code that I can now proceed to flesh out one piece at a time.

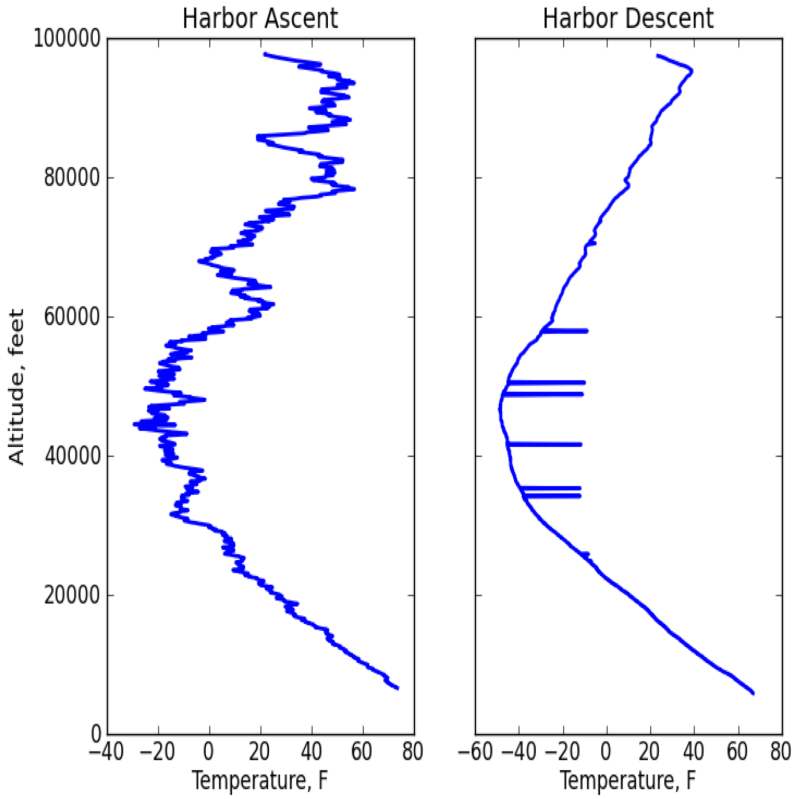
Now the difficult part. You need to write functions for each of those shells that perform the proper tasks. Some tips:

1. Don't forget your top down design! For each function of the program, take a moment to write down a clear statement of what the code will do, what the inputs and outputs are, etc. This will save you a lot of time.
2. Reuse your old code! You've already got some code that reads in files. Cannibalize it, re-write it, and re-use it for this purpose. In fact, if you took a little time now, you could generalize your file reader to work with nearly any file, and would always have that in your toolkit!
3. Web references - and your textbook - are your friends. For example, note that one of your files is a *comma separated value* (csv) file rather than space delimited. You can still use the *split* method from before, but you will have to change it. I would guess it is as simple and using *split(",")* instead of *split()* but don't take my word for it, look it up!
4. Recall the way python returns values from functions. You can return more than one value at a time, so the way these functions are used should give you some clues as to what values you have to return.
5. Note that I am separating the "Up" and "Down" portions of the flight. How will you do that? Keep in mind that the balloon goes up until it doesn't.
6. Make frequent checks of your progress. For example, an excellent way to see if



your first two functions work is to plot that data right away. It will look something like the figure above.

7. Finally, just to give you a hand, I went ahead and wrote the plotting algorithm for the final data. I did this because when atmospheric scientists plot temperature vs. altitude, they do it in a very unique way, switching the x and y axes, and will look like the figure here:



The following code may help.

```
def plotAllFigs(display):

    plot.figure()
    plot.subplot(2,1,1)
    plot.plot(wxTimes, wxTemperatures,linewidth=2.0)
    plot.xlim(0,2.25)
    plot.title("Harbor Flight Data")
    plot.ylabel("Temperature, F")

    plot.subplot(2,1,2)
    plot.plot(gpsTimes, gpsAltitudes,linewidth=2.0)
    plot.xlim(0,2.25)
    plot.ylabel("Altitude, ft")
    plot.xlabel("Mission Elapsed Time, Hours")

    if display == "save":
        plot.savefig("fig2.2.1.png")
```

```
        elif display == "show":
            plot.show()
        else:
            print "Unrecognized output, ", display

        plot.figure()
        plot.subplot(1,2,1)
        plot.title("Harbor Ascent")
        plot.plot(wxCorrelatedTemperaturesUp,
    wxCorrelatedAltitudesUp,linewidth=2.0)
        plot.ylabel("Altitude, feet")
        plot.xlabel("Temperature, F")
        plot.ylim([0,100000])
        plot.subplot(1,2,2).set_yticklabels([])
        plot.title("Harbor Descent")
        plot.plot(wxCorrelatedTemperaturesDown,
    wxCorrelatedAltitudesDown,linewidth=2.0)
        plot.xlabel("Temperature, F")
        plot.ylim([0,100000])

        if display == "save":
            plot.savefig("fig2.2.2.png")
        elif display == "show":
            plot.show()
        else:
            print "Unrecognized output, ", display
```

8. Test and refine: As always.

What to hand in:

1. Generate a complete code that saves the plots above as "raw.png" for the raw data, and "correlated.png" for the correlated data.
2. In your submission, comment on the differences between the ascent and descent temperatures. What could have caused these differences? Hint: and examination of the raw data, especially the GPS data, may help.
3. Also, generate and examine plots for the 2nd and 3rd temperature measurements. When you email me the code, comment on any anomalies. What possible causes - location of sensors, sensor performance, environmental conditions - could cause such differences.
4. **Extra Credit** - Do you see those little data dropouts on the right hand plot (the spikes that go to zero)? Those are bad data. *Without editing the text file* design a way to get rid of those types of data dropouts to clean up the output. 25% extra credit if you do so!