Computer Lab 9: Vostok ice core

Climate Data Analysis, ATS 301, Fall 2018

As before, we start by specifying that we want plots to be displayed inside the Jupyter Notebook, and load the modules we'll need.

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats

# Set default figure size
# This is so I don't have to keep specifying figsize later.
plt.rcParams['figure.figsize'] = (8.0, 6.0)
```

1. Reading in and plotting data

The data are in the `/data/ATS_301/Data/Vostok/` folder.

Look at one of the data files (open it from within Jupyter).

What type of files are these? What are the separators between values?

Use the `np.genfromtxt` command to read the data from the CO_2 file into an array. Do we need to specify a delimiter? We'll need to specify the `skip_header` values again. Unfortunately, they vary among the different files. Use the provided `skip_header` values in the following `np.genfromtxt` commands (so you don't have to determine these yourself).

In [ ]:
```
help(np.genfromtxt)
```

In [ ]:
```
co2=np.genfromtxt("/data/ATS_301/Data/Vostok/co2nat.txt", skip_header=31)
```

In [ ]:
```
print(co2)
```

Verify that the data starts on the correct line by comparing this output with the original data file.

Look at the header for this file. Which columns correspond to which variables? For future reference, I like to save the info about each column in my notebook, so I don't have to go back to the file:

CO2 data: file"co2nat.txt"

Column 1: Gas Age as deduced from the Barnola et al. (1991) model

Column 2: CO2 concentration (ppmv).

Fill in the correct subscripts so you can use more more intuitive variable names for the rest of the lab/homework.

In [ ]:
```
# Fill in correct subscripts!

co2_age = co2[***,***]
co2_conc  = co2[***,***]
```

In [ ]:
```
#
```

Plot the CO_2 concentration as a function of gas age.

In [ ]:
```
# Your code here
```

The x tick labels look a little crowded. One option is to plot age in 1000's of years by dividing the age variable by 1000 and adjusting the plot labels accordingly.

In [ ]:
```
plt.figure(figsize=[8,6])
plt.plot(co2_age/1000,co2_conc)
plt.xlabel('Thousands of Years Before Present')
plt.ylabel('CO$_2$ (ppm)')
plt.show()
```

Next, read in the rest of the data. I've already chosen the correct header lengths.

Look at the files (using jupyter), and determine what each column represents. Note that every file is different!

In [ ]:
```
deut=np.genfromtxt("/data/ATS_301/Data/Vostok/deutnat.txt", skip_header=51)

# Fill in values as appropriate
deut_age = deut[XXXXX,XXXXX]
temp_diff=deut[XXXX,XXXXX]
deut_content=deut[XXXXX,XXXX]

o18=np.genfromtxt("/data/ATS_301/Data/Vostok/o18nat.txt", skip_header=31)
# Fill in values as appropriate
o18_age=o18[XXXX,XXXXX]


dust=np.genfromtxt("/data/ATS_301/Data/Vostok/dustnat.txt", skip_header=31)

na=np.genfromtxt("/data/ATS_301/Data/Vostok/nanat.txt", skip_header=31)

ch4=np.genfromtxt("/data/ATS_301/Data/Vostok/ch4nat.txt", skip_header=31)
```

In [ ]:
```
#
```

Save the column information from these files here for future reference.

XXXXXXXXXXXXXXXXXXXXX

We would like to plot **both** temperature difference and CO_2 versus time **on the same plot**. However, they do not have the same y-axis. To do this, we need to use the `plt.subplots` function (note the "s" on the end, as opposed to the `plt.subplot` function we used to plot multiple figures at once), along with two sets of axes.

Here is an example to get you started. Note that `deut_age`, `deut_temp_diff`, `co2_age`, and `co2_conc` need to have already been assigned.

In [ ]:
```
#Create a new figure and define the first axis to be ax1.
fig, ax1 = plt.subplots(figsize=(10,6))

# Plot temperature on ax1 (the left axis), with a blue line, no markers
ax1.plot(deut_age/1000,temp_diff, 'b-')

# Add the xlabel to be used for both subplots. Leave it black, since both subplots
use it.
ax1.set_xlabel('Time before present (1000 years)')

# Make the y-axis label and tick labels match the line color.
ax1.set_ylabel('*** Fill in correct label (units) *****', color='b')
ax1.tick_params('y', colors='b')  # 'y' indicates we are modifying the ticks on th
e y axis.

# Create a new axis that uses the same x axis as the first one
ax2 = ax1.twinx()

# Plot co2 on ax2, with a red line, and dots
ax2.plot(co2_age/1000,co2_conc, 'r.-')

# Make the y-axis label and tick labels match the line color.
ax2.set_ylabel('*** Fill in correct label (units) *****', color='r')
ax2.tick_params('y', colors='r')

plt.show()
```

Play around with the colors, line/marker, and font size settings to make the plot easy to read, depending on the particular variables you are plotting.

I recommend using a different color for each variable consistently throughout the homework.

Another way you can look at two variables together is to plot them on top of each other as subplots, with the same x axis (`sharex=True`). Here's an example:

In [ ]:
```
# 2 indicates two rows, 1 indicates 1 column
fig, (ax1, ax2)= plt.subplots(2,1,sharex=True,figsize=(10,6))

ax1.plot(deut_age/1000,temp_diff, 'b-')
ax1.set_ylabel('*** Fill in correct label (units) *****', color='b')
ax1.tick_params('y', colors='b')

# You can add lines at various locations to highlight the same time in all the dat
a.
# This is optional.
ax1.axvline(x=128,linewidth=4, color='pink',alpha=0.5)

ax2.plot(co2_age/1000,co2_conc, 'r.-')
ax2.set_ylabel('*** Fill in correct label (units) *****',color='r')
ax2.tick_params('y', colors='r')
ax2.set_xlabel('Time before present (1000 years)')

# Use same x, linewidth, etc. as before. (Again, optional)
ax2.axvline(x=128,linewidth=4, color='pink',alpha=0.5)

# Decrease the horizontal space between plots.
plt.subplots_adjust(hspace=0.1) # default is 0.2

plt.show()
```

For the homework, you will plot each of the variables (CO_2 concentration, methane, dust, sodium, delta O-18) as a function of time, with the Vostok temperature difference on the same axes (as in the first example). [5 figures total]

Note: if a variable is anti-correlated with temperature, you may want to plot -variable to better show the relationship. (Be sure to indicate this in the y axis label if you do this.)

Play around with color, line, and marker styles to highlight relationships between variables.

Then, plot all 6 variables (CO_2 concentration, methane, dust, sodium, delta O-18, **and temperature difference**) in plots stacked on top of each other like the second sample and answer the questions in part 1.

2. Ice age versus gas age

Next, read in the data from gt4nat.txt. Since the first ~100 m does not have a value in the third column, we'll include those lines in the `skip_header` count.

In [ ]:
```
gt4=np.genfromtxt("/data/ATS_301/Data/Vostok/gt4nat.txt", skip_header=129)
```

Look at the file, verify that the first row in our variable is correct, and same the column info.

In [ ]:
```
print(gt4[0,:])
```

Step through the parts of Q2 in HW5.

3. Interpolation, correlations, and linear regression

To determine correlations between two variables and perform linear regressions, the both arrays of variable data must have matching corresponding time values. Based on the plots and text files, we see that they all have different values in the time column. Thus, we must interpolate one variable onto the time values of the other.

Generally, you want to interpolate from the high resolution to the low resolution.

The function we will use is `np.interp` :

In [ ]:
```
help(np.interp)
```

Start by comparing CO_2 and temperature.

Which variable should we interpolate to the other time values?

In [ ]:
```
temp_diff_interp_to_co2_age=np.interp(co2_age,deut_age,temp_diff)
```

This gives us the temperature difference interpolated to the times used for the CO_2 concentation.

Verify the size of the array, and plot the original and interpreted data for comparison.

In [ ]:
```
print(np.shape(temp_diff))
print(np.shape(co2_age))
```

In [ ]:
```
# Which shape should this have?
print(np.shape(temp_diff_interp_to_co2_age))
```

In [ ]:
```
plt.figure(figsize=[8,6])
plt.plot(deut_age,temp_diff)
plt.plot(co2_age,temp_diff_interp_to_co2_age,'k.')
```

Be sure to add labels, etc.

Calculate correlations

Now that we have two variable arrays of the same size, we can use the `stats.pearsonr` and `stats.spearmanr` functions to calculate the correlation between these two variables. Do the two different methods produce similar results?

In [ ]:
```
stats.pearsonr(temp_diff_interp_to_co2_age,co2_conc)
```

The correlation calculation does not depend on the order of the variables. Try the command with the variables reversed:

In [ ]:
```
stats.pearsonr(co2_conc, temp_diff_interp_to_co2_age)
```

In [ ]:
```
stats.spearmanr(temp_diff_interp_to_co2_age,co2_conc)
```

Gives us a very similar result; very reassuring!

Let's see what happens when we plot the two variables against each other.

Which variable should be `x`, and which `y`? (This especially matters when doing line fitting.) That depends on whether you think one determines the other. Does temperature diff determine CO_2 concentration? Does CO_2 concentration determine temperature diff?

For now, let's start by plotting CO_2 as a function of temperature difference.

We can make a scatterplot by specifying only dots as markers (no lines).

In [ ]:
```
plt.plot(temp_diff_interp_to_co2_age,co2_conc,'.')
```

Linear regression

Perform a linear regression:

In [ ]:
```
slope, intercept, r_value, p_value, std_err = stats.linregress(temp_diff_interp_to
_co2_age,co2_conc)
print(slope, intercept, r_value, p_value, std_err)
```

The r_value and p_value match what the Pearson's test function gave us. They should, since the linear regression calculation method uses the Pearson's correlation.

Plot the regression line on the scatter plot.

You can do something like we did earlier (e.g., in HW 1), or you can add some shading around the line to indicate plus and minus one standard error in the slope. This is a little more challenging. The `plt.fill_between` function needs to have the data in increasing x (whereas our temperature doesn't increase monotonically with time), so we create a new array of x values we want to determine the regression line for.

In [ ]:
```
# Create a 1-D array that spans the range of temp diffs, with 100 values
x_values=np.linspace(-9,1,100)

# Create the regression line values
regress_line=intercept+slope*x_values

# Create two arrays with the max and min, respectively,
# values for +/- on standard error in the slope.
regress_line_max=intercept+(slope+std_err)*x_values
regress_line_min=intercept+(slope-std_err)*x_values

# plot the data
plt.figure(figsize=[8,6])
plt.plot(temp_diff_interp_to_co2_age,co2_conc,'.')
# plot the regression line
plt.plot(x_values,regress_line)

# Add the fill
plt.fill_between(x_values,regress_line_min,regress_line_max,color="lightblue")
```

In [ ]:
```
#↔
```

The fact that the shading is so small around the line is a visual demonstration of the statistical significance of the non-zero slope result.

Maybe the relationship isn't really linear, though. We can try some other polynomials to see if they work better.

This function is `np.polyfit`, where the first two arguments are the two arrays, and the third is the degree of the polynomial (1 = line, 2 = parabula, etc.)

Using a degree of 1 should give us the same thing as `stats.linregress`.

In [ ]:
```
np.polyfit(temp_diff_interp_to_co2_age,co2_conc,1)
```

Now try degrees of 2 and 3. (Degree of 2 example below.) Save the fitting coefficients so we can use them to plot. **Note that highest power coefficient is first**.

In [ ]:
```
#fitting parabula
fit_params_2=np.polyfit(temp_diff_interp_to_co2_age,co2_conc,2)
print(fit_params_2)
```

Perform the calculation for a degree of 3. Then plot the 3 fits (degrees=1,2,3) on the scatterplot. Do not add shading for these higher fits.

The next cell adds the first polyfit line, as an example.

In [ ]:
```
# Create a 1-D array that spans the range of temp diffs, with 100 values
x_values=np.linspace(-9,1,100)

# Create the regression line values
regress_line=intercept+slope*x_values
regress_line2=fit_params_2[2]+fit_params_2[1]*x_values+fit_params_2[0]*x_values**2

# Create two arrays with the max and min, respectively,
# values for +/- on standard error in the slope.
regress_line_max=intercept+(slope+std_err)*x_values
regress_line_min=intercept+(slope-std_err)*x_values

# plot the data
plt.figure(figsize=[8,6])
plt.plot(temp_diff_interp_to_co2_age,co2_conc,'.')
# plot the regression line
plt.plot(x_values,regress_line)

# Add the fill
plt.fill_between(x_values,regress_line_min,regress_line_max,color="lightblue")

# Add the 2nd degree regression line
plt.plot(x_values,regress_line2,'r.')
```

In [ ]:
```
#↔
```

Do you think it would be useful to look at higher-degree fits?

Next, re-do the linear regression, this time fitting the temperature difference data to CO_2.

In [ ]:
```
slope1, intercept1, r_value1, p_value1, std_err1 = stats.linregress(co2_conc,temp_
diff_interp_to_co2_age)
print(slope1, intercept1, r_value1, p_value1, std_err1)
```

Compare r_values and p_values. Are they the same?

However, we might initially expect the slopes to be inverses of each other. Are they?

In [ ]:
```
1/slope1
```

Why not?

We really should use a version of linear regression which assumes there is an error in x value as well. But there is no easy way to do this in Python (that I know of) with the packages we have.

Perform the 2-degree and 3-degree polyfits. Do plots as before.

4. Perform the above steps (Q3) with two different variables, and compare with your partners.

See instructions in the homework.

5. [Optional] Calculating Vostok temperature difference from δD

Use the equation from the paper to calculate ΔT from the isotope changes.

$\Delta T_I = (\Delta\delta D_{ice} - 8\Delta\delta^{18}O_{sw})/9$

with $\Delta T_I = 0.67\Delta T_s$.

For $\Delta\delta D_{ice}$, we'll use the differences between the δD_{ice} in the Vostok record and the present-day δD_{ice} (using the first (latest) values from the Vostok data).

For $\Delta\delta^{18}O_{sw}$, we don't have the globally averaged sea water values. But assuming that the time series of changes in $\delta^{18}O_{sw}$ are similar for the globe and the Vostok ice core will get us pretty close. Also, we'll assume that the present-day value is 0.

The only difficulty here is that the ^{18}O record has different ages. So we'll need to interpolate this data to the deuterium ages (found in the deut file).

In [ ]:
```
### Your code here
```

Verify that the size of the resulting array is correct and the data looks reasonable.

In [ ]:
```
print(np.shape(deut_content_interp_to_o18_age))
print(np.shape(deut_age))
print(np.shape(o18_age))
```

In [ ]:
```
plt.plot(deut_age,deut_content)
plt.plot(o18_age,deut_content_interp_to_o18_age,'k.')
```

Now, you can calculate the temperature difference ΔT_I, as well as ΔT_s.

Plot them both on the same plot as the temperature difference given in the deut file.

In [ ]:
```
# Use the equation to calculate the temperature difference↔
```

In [ ]:
```

```