Synoptic Meteorology I: Assignment #2

Due: 25 September 2018

Learning Objectives: Learn how to install additional modules to extend the functionality of the Python language. Learn how one such module can be used to complete a common calculation.

Although Anaconda Python comes with over 200 modules pre-installed, there are many additional modules available, including some that are particularly useful for the atmospheric sciences. In this assignment, we will use the Anaconda Navigator package manager to install several modules, then use Spyder to load one of these modules and use it to convert temperature data between units.

Begin by launching Anaconda Navigator using the desktop shortcut you created in Assignment 1. Next, click on 'Environments' on the left-hand side of the Navigator window. The only source of modules that Navigator knows about at this time is the 'base' source maintained by Anaconda, but the atmospheric science modules we need are located in a different source called conda-forge. To add this source, click on 'Add…', type conda-forge, hit Return, and click on 'Update Channels.'

After you have added the conda-forge module source (or repository, in programming syntax), you can now select and install additional modules. We want to install three modules: metpy, siphon, and cartopy. To do so, type the name of the first module in the top-center search bar (just right of the 'Update index…' button), check the box to the left of its entry, then repeat for the other two modules. Next, click the 'Apply' button at lower right. These modules will require several other modules be installed to support their functionality, but Anaconda Navigator will determine which ones are needed for you. Once it has done this, begin the installation. You can track its progress in the narrow space at the bottom of the Anaconda Navigator window; once there is no longer text in this space (generally after 1-2 minutes), the installation is complete.

Once you have installed these modules, click on 'Home' on the left side of the Navigator window, then launch Spyder. Use the console window to import the metpy module. (Refer to Assignment #1 if you need refreshers on how to launch Spyder or import modules.)

In the remainder of the assignment, we will work with several of metpy's calculation functions to compute various atmospheric quantities and convert between units. To help us do so, let's obtain some atmospheric data. The University of Wyoming maintains an excellent upper-air data archive:

http://weather.uwyo.edu/upperair/sounding.html

Open this web page in your favorite web browser. Change the Year and Month to 2018 and Aug. Change the From and To both to 04/00Z. Once done, click on GRB. A new tab will appear with a text display of the 0000 UTC 4 August 2018 Green Bay, WI upper-air observation. Declare three variables in Python (called p, t, and q) with the pressure, temperature, and mixing ratio values from the first row in which all columns contain data.

One of metpy's core features is the ability to work with common atmospheric units. The variables you defined in the previous step are just numbers to Python, but metpy allows for units to be added to their definitions. This is done through the `metpy.units.units()` function, i.e.,

```
p = p * metpy.units.units('hPa')
```

would add the units of hectopascals (hPa, equivalent to millbars, or mb) to the existing `p` variable. Similar commands can be used to append units such as g kg^{-1} (`g/kg`) or °C (`degC`), among others, to other variables. Do this for all three variables you previously initialized.

Another of metpy's core features is the ability to convert between atmospheric units without you having to do the calculation yourself. This is done via the `to()` attribute of any variable to which you have previous appended units, i.e.,

```
tf = t.to('degF')
```

would convert the units of `t` (which we previously assigned as °C) to °F. Do this for temperature.

Finally, the last of metpy's core features that we will examine in this assignment is the ability to compute derived quantities. However, this relies on functionality that is not automatically loaded with the metpy module. Instead, we have to load this piece of the metpy module separately using another import statement:

```
import metpy.calc as mpcalc
```

You'll frequently encounter this sort of syntax, where a piece of a module is imported and given a shorter name, in Python. Here, instead of referring to `metpy.calc` all the time, we can just use the new `mpcalc` alias in its place.

Once this module has been loaded, we can use it to compute derived quantities. For instance, given pressure and mixing ratio, metpy can compute the vapor pressure, i.e.,

```
e = mpcalc.vapor_pressure(p, q)
```

where `p` and `q` are as defined before, with units. If you were to then print `e`, you would find that the output units are g hPa kg^{-1}, which are kind of wonky. We can use metpy to get rid of g kg^{-1}, leaving only hPa, by using the `to()` attribute described above on the new variable `e`.

Given vapor pressure, metpy's can also compute dew point temperature, i.e.,

td = mpcalc.dewpoint(e)

The default units from this calculation are °C, but you can convert these to °F if desired. A good sanity check on this calculation is to compare it to the University of Wyoming data you began with in this assignment – does the dew point temperature you calculated match the observed value?

For this assignment, you are to obtain the GRB sounding data described above, append units to the data, compute vapor pressure and dew point temperature, and convert both temperature and dew point temperature to °F. Once done, print the resulting data to screen, then use Python to write out a string with your name and the then-current date and time. Take a screenshot and either print it or submit it via e-mail before the start of the next class.

If interested, a complete list of metpy calculation routines (with some examples) is available online at: https://unidata.github.io/MetPy/latest/api/generated/metpy.calc.html#module-metpy.calc.