Week4-ThicknessExample

August 25, 2022

Synoptic Meteorology I: Thickness Example

In our lecture notes, we stated and graphically demonstrated that an area of low pressure at the surface found within a warm air column disappears quickly with height. However, the demonstration relied on a hypothetical example and not actual meteorological data. This Jupyter Notebook uses meteorological data from a WRF-ARW simulation of Hurricane Matthew (2016) to demonstrate this principle for a real-world event.

Data Acquisition and Plot Generation

We start by importing the needed modules. These are drawn from four packages - netCDF4, matplotlib, numpy, and wrf (short for wrf-python). We do not need to load cartopy because there is no mapping involved.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.cm import get_cmap
from matplotlib.ticker import (NullFormatter, ScalarFormatter)
from netCDF4 import Dataset
from wrf import to_np, getvar, CoordPair, vertcross
```

Open the desired WRF-ARW model output file, which in this instance contains data for only a single output time. This is stored in our JupyterHub's community shared_notebooks folder:

[2]:
```
ncfile = Dataset("/srv/data/shared_notebooks/DataWrangling/wrf-python/
 ↪wrfout_d01_2016-10-08_00:00:00")
```

Extract the 3-D model pressure, wind speed, and virtual temperature. The first is in hPa by default; the middle is specified in kt; the last is specified in K. All are wrf-python derived variables.

The uvmet_wspd_dir function returns both speed and direction relative to the Earth. The wind speed is the 0th element of the first dimension and the wind direction is the 1th element of the first dimension. The second dimension is the 3-D data, which we obtain all of.

[3]:
```
p = getvar(ncfile, "pressure")
wspd =  getvar(ncfile, "uvmet_wspd_wdir", units="kt")[0,:]
tv = getvar(ncfile, "tv", units="K")
```

As we did with our hypothetical example in the lecture notes, we will use a vertical cross-section across the simulated Hurricane Matthew to assess our statement. We start by using the wrf-python CoordPair helper function to define the vertical cross-section's

start and end points. We could theoretically use metpy's vertical cross-section functionality (https://unidata.github.io/MetPy/latest/examples/cross_section.html), but we stick to wrf-python's functionality to avoid having to load another package and mix syntax. The points specified here were determined manually after looking at other simulation output for Hurricane Matthew at this time.

[4]:
```
start_point = CoordPair(lat=32.5, lon=-84.0)
end_point = CoordPair(lat=28.0, lon=-78.5)
```

Next, we interpolate our model data to the vertical cross-section using wrf-python's vertcross function. The two commands are nearly identical except for the latlon keyword argument. In the first example, this is set to True so that the lat/lon points along the cross-section axis will be stored with wspd_cross. In the second example, this is set to False so that these points are not stored with theta_cross. Since the cross-section axes are identical, we only need the one set of lat/lon points.

[5]:
```
wspd_cross = vertcross(wspd, p, wrfin=ncfile, start_point=start_point,
                       end_point=end_point, latlon=True, meta=True)
tv_cross = vertcross(tv, p, wrfin=ncfile, start_point=start_point,
                       end_point=end_point, latlon=False, meta=True)
```

We next compute the average virtual temperature at each vertical level along the cross-section's axis, then subtract it from the virtual temperature along the cross-section's axis. The latter gives us the anomalous virtual temperature, which illustrates locations at which the virtual temperature is relatively high or low.

The tv_cross data is an xarray DataArray, so we can use the DataArray .mean() function to compute the average. The tv_cross DataArray has two dimensions (which we can obtain by printing it out), vertical and cross_line_idx. It is the latter along which we compute the average.

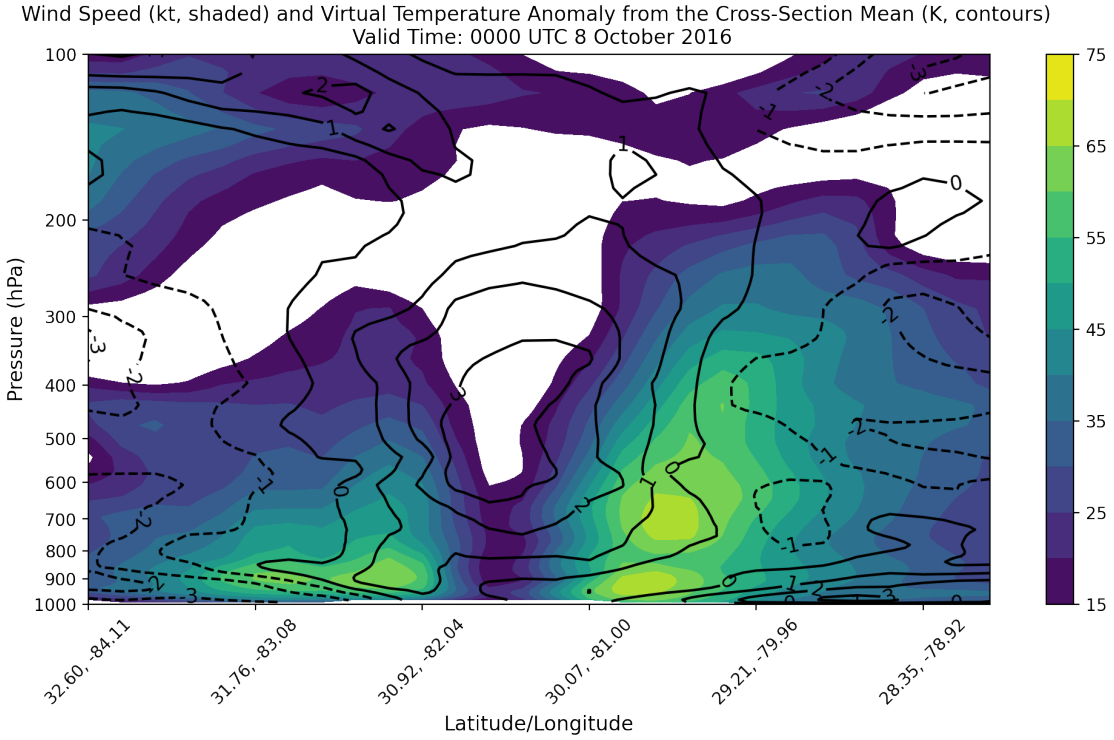
[6]:
```
tv_cross_mean = tv_cross.mean(dim='cross_line_idx')
tv_cross_anom = tv_cross-tv_cross_mean
```

The remainder of the plot-generation code is contained in a single code block below. This is due to a Python quirk; a figure is generated before we add any data to it if we try to break the code up into separate code blocks. Please see the comment blocks below to interpret the code.

[7]:
```
# Create the figure instance (12" wide by 6" tall,
# 200 dots per inch), then establish the figure's axes.
fig = plt.figure(figsize=(12,6), dpi=200.)
ax = plt.axes()

# We use contourf to plot the cross-section data, which
# are stored in the wspd_cross variable defined at the
# end of the previous code block. In addition to the data,
# this variable has two coordinate dimensions of relevance:
# xy_loc, which contains the lat/lon points along the cross-
# section, and vertical, which contains the vertical levels
# for the vertical cross-section.
```

```
# The x-axis is a 2-D location. When plotting, however, we
# can only pass in one dimension. We handle this by passing
# in a 1-D array of values from 0 -> N, where N is the number
# of locations along our vertical cross-section. We later loop
# over the location coordinates to get lat/lon information for
# labeling the x-axis.
# The y-axis is pressure. We can get this from wspd_cross's
# vertical coordinate.
# All fields are converted from their default xarray fields
# to numpy arrays for ease of plotting. The numpy arrays do
# not have descriptive metadata and thus are well-suited for
# basic plotting operations such as those here.
# We specify shading levels from 15 (inclusive) to 80
# (exclusive) by 5 kt.
# We specify that the plot should use the viridis colormap.
# More info on colormaps:
# https://matplotlib.org/stable/tutorials/colors/colormaps.html
coord_pairs = to_np(wspd_cross.coords["xy_loc"])
wspd_contours = ax.contourf(np.arange(coord_pairs.
 ↪shape[0]),to_np(wspd_cross["vertical"]),
                             to_np(wspd_cross),np.arange(15.,80.,5.),␣
 ↪cmap=get_cmap("viridis"))

# Add a colorbar.
plt.colorbar(wspd_contours, ax=ax)

# Add black contours for virtual temperature anomaly. As with wind
# speed, we pass a 1-D array from 0 -> N for the x-dimension,
# then pass in the vertical levels for the y-dimension. We
# specify vertical levels from -5 K (inclusive) to +5 K
# (inclusive) by 1 K. Not specifying a line type defaults to
# solid lines for positive values and dashed lines for negative values.
tv_contours = ax.contour(np.arange(coord_pairs.
 ↪shape[0]),to_np(tv_cross_anom["vertical"]),
                            to_np(tv_cross_anom),np.arange(-5.,5.01,1.
 ↪0),colors='black')
plt.clabel(tv_contours, inline=1, fontsize=12, fmt="%i")

# This set of code structures our x-axis ticks and their labels.
# It relies on the coord_pairs set of x/y, lat/lon coordinate
# information defined a few lines above. First, we create an
# array from 0 -> N to generically define the axis tick marks.
# Next, we loop over the coord_pairs variable to extract out
# the lat/lon information (using its latlon_str helper function,
# defined in wrf-python's CoordPairs module). These positions,
# which will end up being our tick labels, are stored to x_labels.
# Finally, we set the tick mark locations and tick labels,
```

```
# where ::5 for each indicates all values from start to end
# (the :: part) by 5. You may need to tweak this depending on
# how long of a cross-section you have. Note that the tick
# labels are rotated slightly for display purposes.
x_ticks = np.arange(coord_pairs.shape[0])
x_labels = [pair.latlon_str(fmt="{:.2f}, {:.2f}")
            for pair in to_np(coord_pairs)]
ax.set_xticks(x_ticks[::5])
ax.set_xticklabels(x_labels[::5], rotation=45, fontsize=10)

# This set of code structures our y-axis ticks and their labels.
# We first set the y-axis to be logarithmic rather than linear.
# Next, we set how the logarithmic axis labels should be structured,
# using scalars rather than powers of 10. Once we have
# done that, we define ten y-axis ticks from 100 to 1000 hPa.
# Finally, we set the y-axis limits - in this case, 1000-100 hPa.
ax.set_yscale('symlog')
ax.yaxis.set_major_formatter(ScalarFormatter())
ax.set_yticks(np.linspace(100, 1000, 10))
ax.set_ylim(1000, 100)

# Set the x-axis and  y-axis labels
ax.set_xlabel("Latitude/Longitude", fontsize=12)
ax.set_ylabel("Pressure (hPa)", fontsize=12)

# Title the plot and then display it. Note the \n
# operate to split the title over two lines.
plt.title("Wind Speed (kt, shaded) and Virtual Temperature Anomaly from the␣
 ↪Cross-Section Mean (K, contours) \n Valid Time: 0000 UTC 8 October 2016")
plt.show()
```



Interpretation

Hurricane Matthew is centered in the above cross-section. The near-surface wind speed exceeds 60 kt along Matthew's northwest side, or to the left of the center along the cross-section, and exceeds 65 kt along Matthew's southeast side, or to the right of the center along the cross-section. The wind speed rapidly decays with increasing altitude or decreasing pressure on both sides, representing an area of low pressure that weakens with increasing altitude.

Concurrently, the virtual temperature anomaly in the vertical column atop Hurricane Matthew exceeds +3 K between 625-325 hPa and is positive throughout the troposphere. This indicates that Hurricane Matthew is located within a column of warm air relative to its surroundings.

Altogether, we have an area of low pressure at the surface within a column of warm air. Consistent with thickness principles, this area of low pressure rapidly decays with increasing altitude, thus providing a real-world example to support what the math and hypothetical example tells us.