5.2

FROM GEMPAK TO METPY: AN UPDATED TOOL FOR TEACHING SYNOPTIC METEOROLOGY (AND MUCH MORE)

Kevin H. Goebbert*, Ryan M. May**, John R. Leeman**
* Valparaiso University, Valparaiso, IN
** UCAR/Unidata, Boulder, CO

1. INTRODUCTION

A critical piece to the study of the atmosphere is the ability to create and analyze weather maps. For decades that primarily meant receiving the National Weather Service (NWS) Digital Facsimile (DIFAX) maps to analyze current conditions as well as forecasts. In the 1980's the GEneral Meteorological PAckage (GEMPAK) was launched through the NWS and made available to the university community through the UCAR/Unidata program. GEMPAK has been a primary tool for research and teaching for the past 25 years or more, but in 2008 the NWS announced it would cease development of the program. Since that time, Unidata has continued to support the legacy soft-ware making needed changes and keeping the decoders updated as new products become avail-able. However, a replacement is needed as the long-term viability of maintaining a software pack-age originally developed in the 1980's becomes increasingly difficult. Additionally, GEMPAK is largely not used outside of the academy limiting is usefulness for preparing students for the modern workforce.

The question has lingered since 2008, what should replace GEMPAK? At the time there were a few options including Grads, NCL, and IDV, but none of those fit the mold of a simplified syntax that had a relatively robust functionality, that could batch process data, produce publication quality graphics, and have a feasible learning curve. For various reasons, all of the "well-known" software packages of the mid to late 2000s came up short. So, the search has continued.

Around this same time, the Python programing language was beginning to be adopted at a greater rate within the scientific community writ large. The adoption provided critical mass to develop tested, fast, and robust modules to support the community and their needs. In addition, these modules were built by the community, for the community, and available for free under various common open-source licenses. This provided a cost-friendly option rather than pay for a license to the Matlab program. Thus, there has been widening support and adoption for Python modules such as Matplotlib, Numpy, and SciPy within the scientific community. Quickly many disciplines developed modules that brought specific functionality to the Python ecosystem that they needed in order to efficiently do their work. However, the atmospheric science community lagged and as such at a slower adoption rate of Python, coupled with the fact that the community still has a large and active Fortran developer community in numerical modelling and enjoyed numerous data analysis and visualization packages that were still running and supported at the time.

Over the last decade, the landscape of computing has continued to evolve. The GEMPAK data format continues to be difficult to maintain but is still being led by Unidata. The scientific Python community has continued to grow, and the number of modules has grown substantially with even greater adoption in the atmospheric science community. In late 2015 Unidata adopted a fledgling Python module called MetPy (May et al. 2017), which originally came into existence to house code used for a couple of graduate students' research and has been actively developing the module with community members over the past three years. In that time there have been 21 of releases that have greatly increased the functionality and usefulness for those interested in synoptic-scale analyses similar to those produced by the legacy software GEMPAK.

This extended abstract is intended to make the case that the Scientific Python-stack and MetPy are the long-term solution to the legacy GEMPAK software package. Additionally, this paper includes examples of Python scripts that use the scientific-stack, including MetPy, to create classic synoptic meteorology graphics that can be used for teaching and research. These examples require the use of MetPy 0.10 or later.

2. Replacing GEMPAK

Since the NWS indicated that they were no longer actively developing GEMPAK or maintaining the decoders to put the vast amounts of meteorological data into the GEMPAK binary format, the need for an alternative solution for creating synoptic-scale analyses has been needed. The requirements to replace GEMPAK:

* *Corresponding Author*: Dr. Kevin H. Goebbert, Valparaiso University, 1809 Chapel Dr., Valparaiso, IN 46383; email: kevin.goebbert@valpo.edu

i. Produce publication quality graphics
ii. Allow for batch operations
iii. Calculate common meteorological variables such as absolute vorticity, isentropic levels, and potential vorticity
iv. Have a reasonable learning curve
v. Ideally have a community development mindset that is not tied specifically to institutional (e.g., governmental) support for development and maintenance.

This is not too tall of an order, but none of the available software programs/packages available when GEMPAK development ceased adequately fulfilled all of these requirements.

3. The Case for MetPy

Many of the other potential options for GEMPAK alternatives are either based on compiled programming languages or are strictly (or realistically) only a graphical user interface (GUI) program. There are also very few programs that can produce modern publication quality graphics in both raster and vector forms. Programs based on compiled languages and libraries are notoriously difficult to compile, maintain, and typically don't have a substantial user-development community. The prospects that some of the already developed tools lasting another 10, 20, or 30+ years was unlikely and would only push the proverbial can down the road for only a little while longer.

In many respects, Python has been a game changer within the scientific community. The base language is relatively easy to learn and those with a Fortran background will benefit from the mathematical operations using the same syntactical structure. Being a full programming language, batch operations are inherently a part of any scripting process one could devise. Python also has a rich set of community-maintained modules that make it possible to obtain many kinds of functionality. There are also graphics tools allowing users to build their own graphical interfaces if they desire.

Despite this, there have been historic limitations to the base language that limited its use within the larger scientific community for quite some time. The limitations of the base language have largely been overcome through the community-driven development to increase speed and performance of operations common to scientific users (e.g., array operations). Additionally, the ecosystem allows for robust testing, software tracking, and easy community feedback to guide module development. It is within this larger ecosystem that the MetPy module began development and reached a tipping point in early 2017 with a growing number of early adopters.

The MetPy module is a community driven development led by software engineers at Unidata, who have long led the development of atmospheric science related software solutions, including GEMPAK, LDM, and IDV. As of January 2019, the calculation functionality available in MetPy or through the Python scientific stack has become quite robust. While full calculation parity to GEMPAK has yet to be achieved, all of the most commonly used calculation and plotting features are available. There are also a number of instances of functionality that goes beyond that which is available in many if not most of the legacy software programs.

Currently the package boasts a total of 82 calculations that can be performed on grids and 17 specific sounding related calculations. There are 9 interpolation schemes ranging from Barnes and Cressman to isentropic vertical interpolation. There are methods to plot surface and upper-level station observations, as well as skew-T log-p diagrams. Additionally, coupled with Matplotlib, publication quality graphics are straightforward to make.

Despite all of the calculations and plotting techniques there are still a few key improvements/developments that are needed to fully replace GEMPAK:

i. Simplified, declarative syntax
ii. Automated calculation solver
iii. Easy read access for surface and upper-air observational data
iv. Full comparison to GEMPAK calculations

All of these improvements are already in the development process and will increasingly make MetPy the most viable successor to GEMPAK that we have had in the past decade. Through the continued support of the atmospheric science community, the module will be set to surpass the functionality of GEMPAK in the near future.

In addition, there is already a wealth of training materials available from Unidata to aid in adoption of MetPy. The materials range from an introduction to the Python language, to examples built into the module itself, to training workshop materials, and an extra examples gallery. If there are missing pieces to training that is not covered by one of these sources, issues can be submitted to their respective GitHub repositories.

4. Conclusion

As the atmospheric science community is about to start the third decade of the 21st century it is time to evolve to the next generation of software for data analysis and visualization. The MetPy module delivers the best compromise of qualities needed for a useful replacement for GEMPAK style analyses, especially those of the synoptic-dynamic genre. With continued community support (through identifying and reporting issues and submitting pull requests) and leadership from the Unidata program, MetPy will be a long-term solution to creating high-quality, publication ready graphics for decades to come.

REFERENCES

May, Ryan M., Arms, S. C., Marsh, P., Bruning, E. and Leeman, J. R., 2017: MetPy: A Python Package for Meteorological Data. Unidata, Accessed 18 December 2018. [Available online at https://github.com/Unidata/MetPy.] doi:10.5065/D6WW7G29.

APPENDIX: Example Scripts

Examples given in the appendix are also available as Python scripts and Jupyter notebooks at https://github.com/kgoebber/synoptic_meteorology and data for the examples are able to be downloaded from http://bergeron.valpo.edu/python/example_data/.

i. 500-hPa Absolute Vorticity
ii. 300 K Isentropic Surface
iii. 250-hPa Baroclinic Potential Vorticity
iv. Skew-T Log-p Diagram
v. GOES 16 Infrared Satellite Image

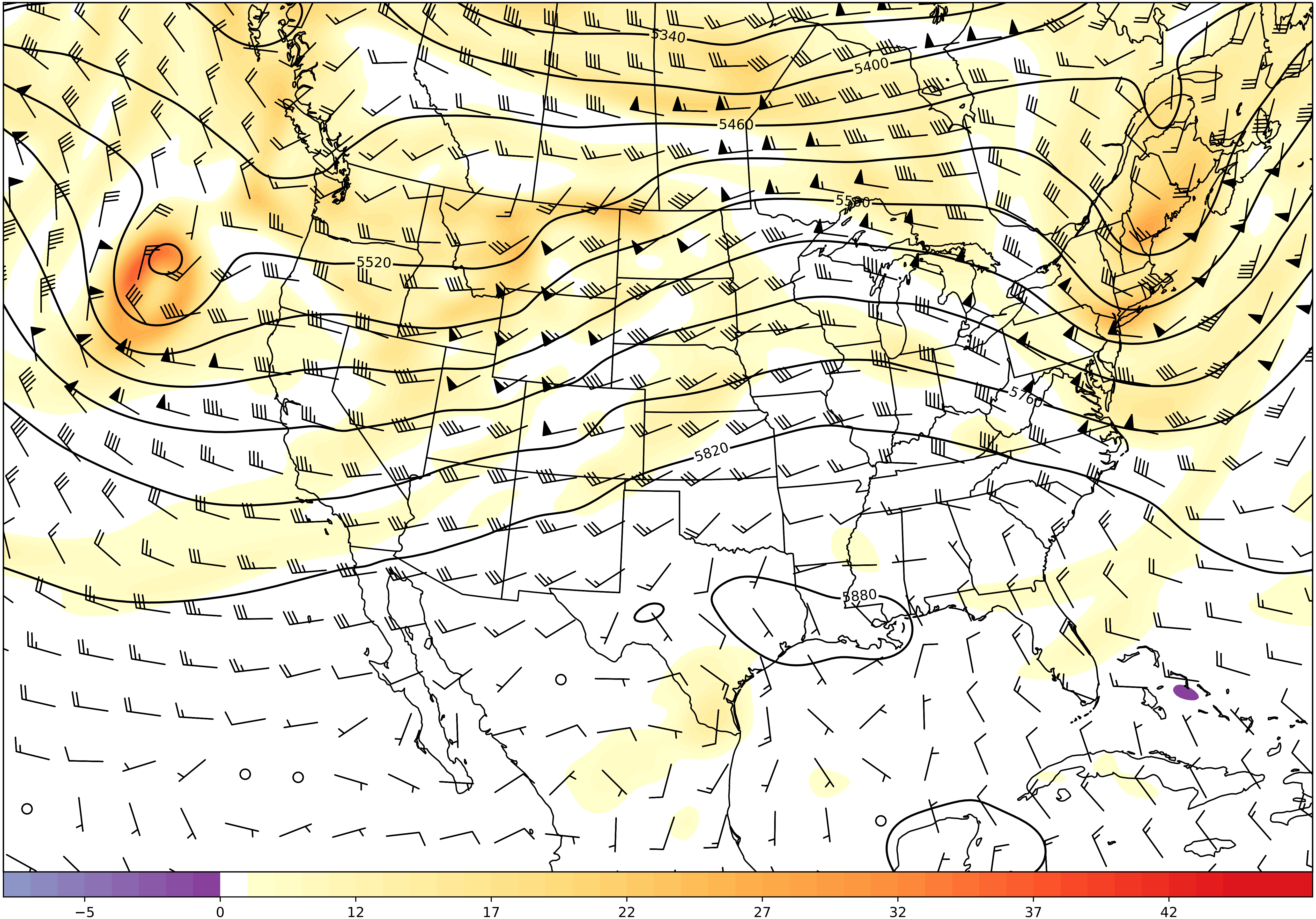
500-hPa Absolute Vorticity Example

```
"""
500 hPa Geopotential Heights, Absolute Vorticity, and Winds
===========================================================

Classic 500-hPa absolute vorticity plot using NAM analysis file.

This example uses example data from the NAM analysis for 12 UTC 31
October 2016 and uses xarray as the main read source with using MetPy to
calculate absolute vorticity and wind speed with geographi plotting
using Cartopy for a CONUS view of the 500-hPa geopotential heights,
absolute vorticity, and wind barbs.

"""


######################################################################
# Import the needed modules.
#

from datetime import datetime

import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt
import metpy.calc as mpcalc
from metpy.units import units
import numpy as np
from scipy.ndimage import gaussian_filter
import xarray as xr

######################################################################
# The following code reads the example data using the xarray open_dataset
# function and prints the coordinate values that are associated with the
# various variables contained within the file.
#

ds = xr.open_dataset('NAM_20161031_1200.nc')
print(ds)


######################################################################
# Data Retrieval
# --------------
#
# This code retrieves the necessary data from the file and completes some
# smoothing of the geopotential height and wind fields using the SciPy
# function gaussian_filter. A nicely formated valid time (vtime) variable
# is also created.
#

# Grab lat/lon values (NAM will be 2D)
lats = ds.lat.data
lons = ds.lon.data

# Select and grab 500-hPa geopotential heights and wind components,
# smooth with gaussian_filter
hght_500 = mpcalc.smooth_n_point(ds.Geopotential_height_isobaric.sel(
                                 isobaric=500).data[0], 9, 50)
uwnd_500 = mpcalc.smooth_n_point(ds['u-component_of_wind_isobaric'].sel(
                                 isobaric=500).data[0], 9, 50) * units('m/s')
vwnd_500 = mpcalc.smooth_n_point(ds['v-component_of_wind_isobaric'].sel(
                                 isobaric=500).data[0], 9, 50) * units('m/s')

# Create a clean datetime object for plotting based on time of Geopotential heights
vtime = datetime.strptime(str(ds.time.data[0].astype('datetime64[ms]')),
                          '%Y-%m-%dT%H:%M:%S.%f')


######################################################################
# MetPy Absolute Vorticity Calculation
# ------------------------------------
#
# This code first uses MetPy to calcualte the grid deltas (sign aware) to
```

```
# use for derivative calculations with the funtcion
# ``lat_lon_grid_deltas()`` and then calculates ``absolute_vorticity()``
# using the wind components, grid deltas, and latitude values.
#

# Calculate grid spacing that is sign aware to use in absolute vorticity calculation
dx, dy = mpcalc.lat_lon_grid_deltas(lons, lats)

# Calculate absolute vorticity from MetPy function
avor_500 = mpcalc.absolute_vorticity(uwnd_500, vwnd_500, dx, dy,
                                     lats * units.degrees, dim_order='yx')


######################################################################
# Map Creation
# ------------
#
# This next set of code creates the plot and draws contours on a Lambert
# Conformal map centered on -100 E longitude. The main view is over the
# CONUS with geopotential heights contoured every 60 m and absolute
# vorticity colorshaded (:math:`*10^5`).
#

# Set up the projection that will be used for plotting
mapcrs = ccrs.LambertConformal(central_longitude=-100, central_latitude=35,
                               standard_parallels=(30, 60))

# Set up the projection of the data; if lat/lon then PlateCarree is what you want
datacrs = ccrs.PlateCarree()

# Start the figure and create plot axes with proper projection
fig = plt.figure(1, figsize=(17, 16))
ax = plt.subplot(111, projection=mapcrs)
ax.set_extent([-130, -72, 20, 55], ccrs.PlateCarree())

# Add geopolitical boundaries for map reference
ax.add_feature(cfeature.COASTLINE.with_scale('50m'))
ax.add_feature(cfeature.STATES.with_scale('50m'))

# Absolute Vorticity colors
# Use two different colormaps from matplotlib and combine into one color set
clevs_500_avor = list(range(-8, 1, 1))+list(range(8, 46, 1))
colors1 = plt.cm.YlOrRd(np.linspace(0, 1, 48))
colors2 = plt.cm.BuPu(np.linspace(0.5, 0.75, 8))
colors = np.vstack((colors2, (1, 1, 1, 1), colors1))

# Plot absolute vorticity values (multiplying by 10^5 to scale appropriately)
cf = ax.contourf(lons, lats, avor_500*1e5, clevs_500_avor, colors=colors,
                 extend='max', transform=datacrs)
plt.colorbar(cf, orientation='horizontal', pad=0, aspect=50, extendrect=True)

# Plot 500-hPa Geopotential Heights in meters
clevs_500_hght = np.arange(0, 8000, 60)
cs = ax.contour(lons, lats, hght_500, clevs_500_hght, colors='black',
                transform=datacrs)
plt.clabel(cs, fmt='%d')

# Set up a 2D slice to reduce the number of wind barbs plotted (every 20th)
wind_slice = (slice(None, None, 20), slice(None, None, 20))
ax.barbs(lons[wind_slice], lats[wind_slice],
         uwnd_500.to('kt')[wind_slice].m, vwnd_500[wind_slice].to('kt').m,
         pivot='middle', color='black', transform=datacrs)

# Plot two titles, one on right and left side
plt.title('500-hPa NAM Geopotential Heights (m), Abs. Vorticity ($s^{-1}$),'
          ' and Wind Barbs (kt)', loc='left')
plt.title('Valid Time: {}'.format(vtime), loc='right')

plt.savefig('500_hPa_Abs_Vorticity.pdf', dpi=150, bbox_inches='tight')
plt.close()
```

500-hPa NAM Geopotential Heights (m), Abs. Vorticity (s^{-1}), and Wind Barbs (kt)

Valid Time: 2016-10-31 12:00:00

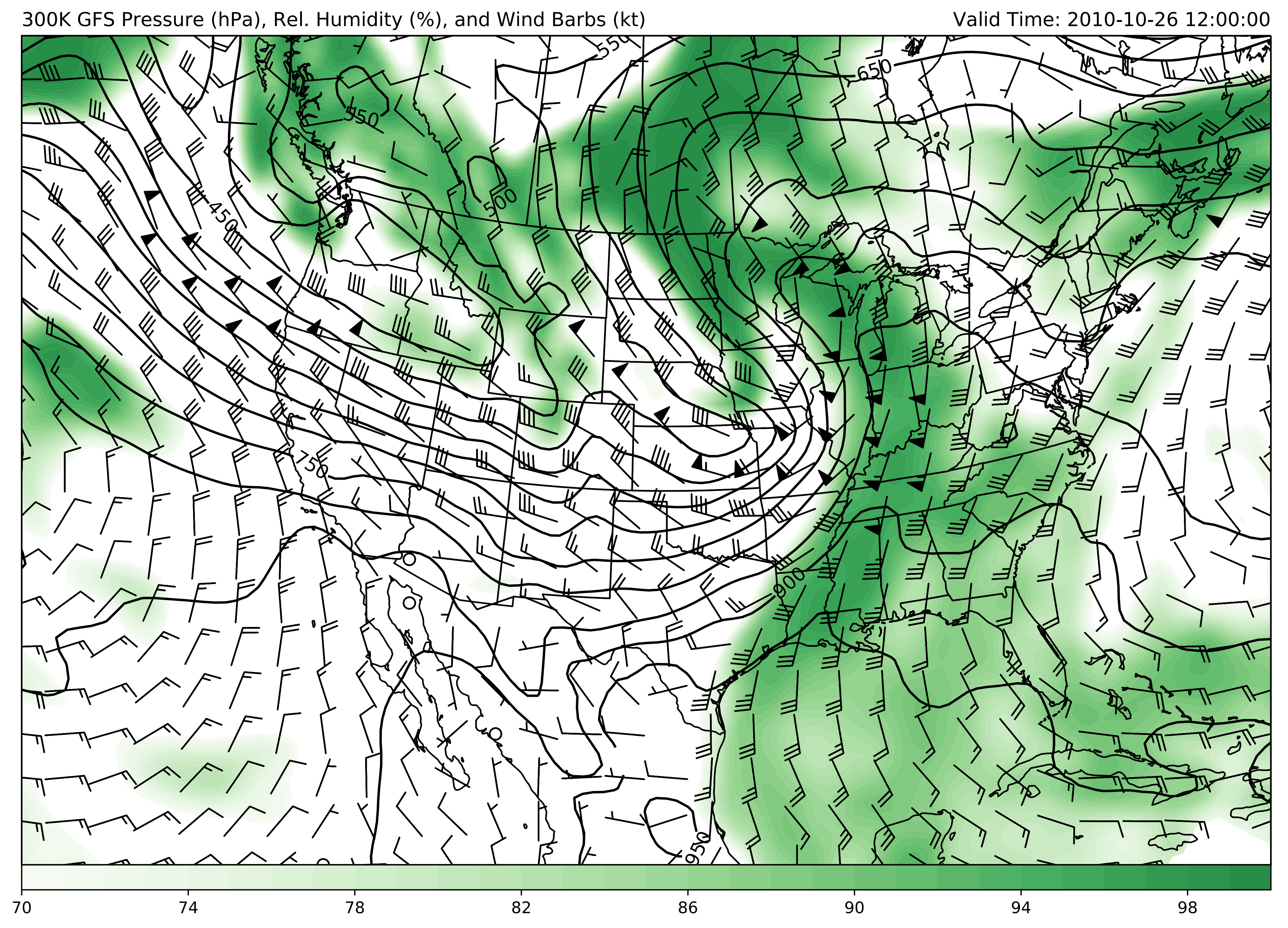


300 K Isentropic Surface Example

```
"""
Isentropic Analysis, Relative Humidity, and Winds
=================================================

Classic isentropic level plot using GFS analysis file.

This example uses example data from the GFS analysis for 12 UTC 31
October 2016 and uses xarray as the main read source with using MetPy to
calculate the isentropic levels and wind speed with geographic plotting
using Cartopy for a CONUS view of the 300 K isentropic surface with
relative humidity, and wind barbs.

"""

from datetime import datetime

import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt
import metpy.calc as mpcalc
from metpy.units import units
import numpy as np
from scipy.ndimage import gaussian_filter
import xarray as xr


######################################################################
# The following code reads the example data using the xarray open_dataset
# function and prints the coordinate values that are associated with the
# various variables contained within the file.
#

ds = xr.open_dataset('GFS_20101026_1200.nc')
print(ds)


######################################################################
# Data Retrieval
# --------------
#
# This code retrieves the necessary data from the file and completes some
# smoothing of the geopotential height and wind fields using the SciPy
# function gaussian_filter. A nicely formated valid time (vtime) variable
# is also created.
#

# Grab lat/lon values (GFS will be 1D)
lat = ds.lat.data
lon = ds.lon.data

# Set subset slice for the geographic extent of data to limit download
lon_slice = slice(400,701)
lat_slice = slice(10,160)

# Subset lat/lon values
lons = lon[lon_slice]
lats = lat[lat_slice]

# Grab the pressure levels and select the data to be imported
# Need all pressure levels for Temperatures, U and V Wind, and Rel. Humidity
# Smooth with the gaussian filter from scipy
pres = ds['isobaric3'].data[:] * units('Pa')
tmpk_var = ds['Temperature_isobaric'].data[0, :, lat_slice, lon_slice]
tmpk = gaussian_filter(tmpk_var, sigma=1.0) * units.K
thta = mpcalc.potential_temperature(pres[:, None, None], tmpk)

uwnd_var = ds['u-component_of_wind_isobaric'].data[0, :, lat_slice, lon_slice]
vwnd_var = ds['v-component_of_wind_isobaric'].data[0, :, lat_slice, lon_slice]
uwnd = gaussian_filter(uwnd_var, sigma=1.0) * units('m/s')
vwnd = gaussian_filter(vwnd_var, sigma=1.0) * units('m/s')

relh_var = ds['Relative_humidity_isobaric'].data[0, :, lat_slice, lon_slice]
relh = gaussian_filter(relh_var, sigma=1.0) * units('percent')
```

```
# Need to do some work to add in missing pressure level for
# Relative Humidity Data - missing 2000 Pa level (index value 1 in tmpk array)
relh2 = np.empty(tmpk.shape)
missing_level = np.empty(relh.shape[1:])
missing_level.fill(np.nan)
relh2[0] = relh[0]
relh2[1] = missing_level
relh2[2:] = relh[1:]

# Create a clean datetime object for plotting based on time of Geopotential heights
vtime = datetime.strptime(str(ds.time.data[0].astype('datetime64[ms]')),
                          '%Y-%m-%dT%H:%M:%S.%f')


######################################################################
# Use MetPy to calcualte multiple isentropic levels from 280 - 300 K
#

isentlevs = list(range(280,331,2)) * units.K
print(isentlevs)
isent_anal = mpcalc.isentropic_interpolation(isentlevs,
                                             pres,
                                             tmpk,
                                             relh2,
                                             uwnd,
                                             vwnd)

# Isolate isentropic variables after interpolation convert winds to knots
isentprs, isentrelh, isentu, isentv = isent_anal
isentu.ito('kt')
isentv.ito('kt')

# Determine the index value of 300 K
ilev = list(isentlevs.m).index(300)


######################################################################
# Map Creation
# ------------
#
# This next set of code creates the plot and draws contours on a Lambert
# Conformal map centered on -100 E longitude. The main view is over the
# CONUS with isentropic map with pressure contoured every 50 hPa and
# relative humidity colorshaded above 70%.
#

# Set up the projection that will be used for plotting
mapcrs = ccrs.LambertConformal(central_longitude=-100, central_latitude=35,
                               standard_parallels=(30, 60))

# Set up the projection of the data; if lat/lon then PlateCarree is what you want
datacrs = ccrs.PlateCarree()

# Start the figure and create plot axes with proper projection
fig = plt.figure(1, figsize=(14,12))
ax = plt.subplot(111, projection=mapcrs)
ax.set_extent([-130, -72, 20, 55], ccrs.PlateCarree())

# Add geopolitical boundaries for map reference
ax.add_feature(cfeature.COASTLINE.with_scale('50m'))
ax.add_feature(cfeature.STATES.with_scale('50m'))

# Plot colorfilled RH >= 70%
clevs_relh = np.arange(70, 101, 1)
cf = ax.contourf(lons, lats, isentrelh[ilev], clevs_relh, cmap=plt.cm.Greens,
                 norm=plt.Normalize(70, 110), transform=datacrs)
plt.colorbar(cf, orientation='horizontal', pad=0, aspect=50)

# Plot isobars every 50 hPa
clevs_pres = np.arange(0, 1100, 50)
cs1 = ax.contour(lons, lats, mpcalc.smooth_n_point(isentprs[ilev], 9), clevs_pres,
                 colors='black', transform=datacrs)
```

```
plt.clabel(cs1, fmt='%d', fontsize='large')

# Plot wind barbs
ax.barbs(lons, lats, isentu[ilev].m, isentv[ilev].m, pivot='middle',
         color='black', regrid_shape=20, transform=datacrs)

# Plot some nice titles on the left and right side of the top of the image
plt.title('{}K GFS Pressure (hPa), Rel. Humidity (%),'
          ' and Wind Barbs (kt)'.format(isentlevs[ilev].m), loc='left')
plt.title('Valid Time: {}'.format(vtime), loc='right')

plt.savefig('300_K_Isentropic.png', dpi=150, bbox_inches='tight')
plt.close()
```



300K GFS Pressure (hPa), Rel. Humidity (%), and Wind Barbs (kt)
Valid Time: 2010-10-26 12:00:00
450
500
550
650
750
900
950
70
74
78
82
86
90
94
98

250-hPa Baroclinic Potential Vorticity Example

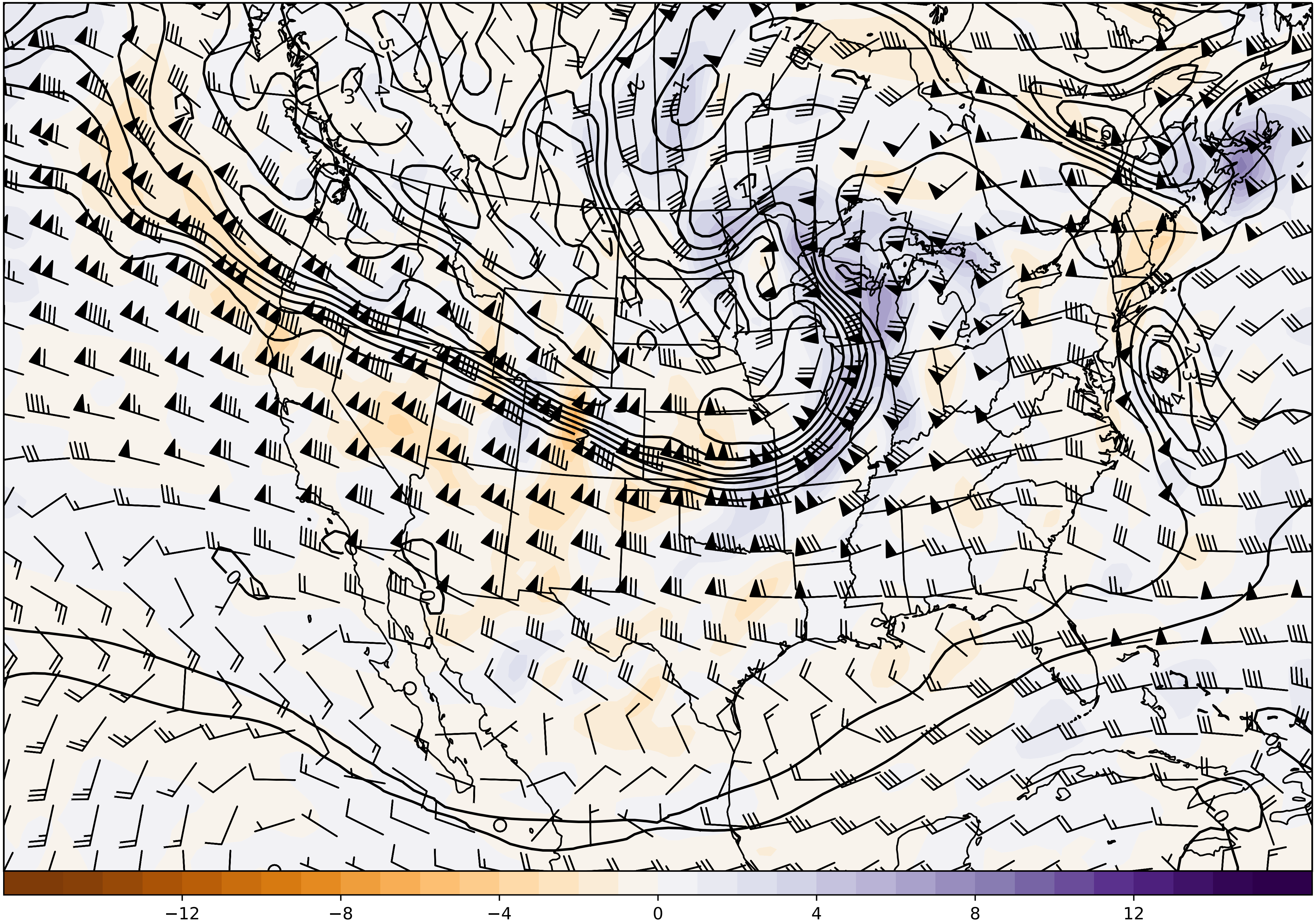
```python
"""
Baroclinic Potential Vorticity Analysis, Divergence, and Winds
==============================================================

Classic baroclinic potential vorticity plot at 250 hPa using GFS
analysis file.

This example uses example data from the GFS analysis for 12 UTC 31
October 2016 and uses xarray as the main read source with using MetPy to
calculate the baroclinic potential vorticity, divergence and wind speed
with geographic plotting using Cartopy for a CONUS view of the 250-hPa
surface with divergence and wind barbs.

"""

from datetime import datetime

import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt
import metpy.calc as mpcalc
from metpy.units import units
import numpy as np
from scipy.ndimage import gaussian_filter
import xarray as xr


######################################################################
# The following code reads the example data using the xarray open_dataset
# function and prints the coordinate values that are associated with the
# various variables contained within the file.
#

ds = xr.open_dataset('GFS_20101026_1200.nc')
print(ds)


######################################################################
# Data Retrieval
# --------------
#
# This code retrieves the necessary data from the file and completes some
# smoothing of the temperature, and wind fields using the SciPy function
# gaussian_filter. A nicely formated valid time (vtime) variable is also
# created.
#

# Grab lat/lon values (GFS will be 1D)
lat = ds.lat.data
lon = ds.lon.data

# Set subset slice for the geographic extent of data to limit download
lon_slice = slice(400,701)
lat_slice = slice(10,160)

# Subset lat/lon values
lons = lon[lon_slice]
lats = lat[lat_slice]

# Grab the pressure levels and select the data to be imported
# Need all pressure levels for Temperatures, U and V Wind, and Rel. Humidity
# Smooth with the gaussian filter from scipy
pres = ds['isobaric3'].data[:] * units('Pa')

tmpk_var = ds['Temperature_isobaric'].data[0, :, lat_slice, lon_slice]
tmpk = gaussian_filter(tmpk_var, sigma=1.0) * units.K
thta = mpcalc.potential_temperature(pres[:, None, None], tmpk)

uwnd_var = ds['u-component_of_wind_isobaric'].data[0, :, lat_slice, lon_slice]
vwnd_var = ds['v-component_of_wind_isobaric'].data[0, :, lat_slice, lon_slice]
uwnd = gaussian_filter(uwnd_var, sigma=1.0) * units('m/s')
vwnd = gaussian_filter(vwnd_var, sigma=1.0) * units('m/s')
```

```
# Create a clean datetime object for plotting based on time of Geopotential heights
vtime = datetime.strptime(str(ds.time.data[0].astype('datetime64[ms]')),
                          '%Y-%m-%dT%H:%M:%S.%f')


######################################################################
# Use MetPy to compute the baroclinic potential vorticity on all isobaric
# levels and other variables
#

# Compute dx and dy spacing for use in vorticity calculation
dx, dy = mpcalc.lat_lon_grid_deltas(lons, lats)

# Comput the PV on all isobaric surfaces
pv = mpcalc.potential_vorticity_baroclinic(thta, pres[:, None, None], uwnd, vwnd,
                                           dx[None, :, :], dy[None, :, :],
                                           lats[None, :, None] * units('degrees'))

# Use MetPy to compute the divergence on the pressure surfaces
div = mpcalc.divergence(uwnd, vwnd, dx[None, :, :], dy[None, :, :], dim_order='yx')

# Find the index value for the 250-hPa surface
i250 = list(pres.m).index(((250 * units('hPa')).to(pres.units)).m)


######################################################################
# Map Creation
# ------------
#
# This next set of code creates the plot and draws contours on a Lambert
# Conformal map centered on -100 E longitude. The main view is over the
# CONUS with isentropic map with pressure contoured every 50 hPa and
# relative humidity colorshaded above 70%.
#

# Set up the projection that will be used for plotting
mapcrs = ccrs.LambertConformal(central_longitude=-100, central_latitude=35,
                               standard_parallels=(30, 60))

# Set up the projection of the data; if lat/lon then PlateCarree is what you want
datacrs = ccrs.PlateCarree()

# Start the figure and create plot axes with proper projection
fig = plt.figure(1, figsize=(14,12))
ax = plt.subplot(111, projection=mapcrs)
ax.set_extent([-130, -72, 20, 55], ccrs.PlateCarree())

# Add geopolitical boundaries for map reference
ax.add_feature(cfeature.COASTLINE.with_scale('50m'))
ax.add_feature(cfeature.STATES.with_scale('50m'))

# Plot the contours of PV at 250 hPa, scaling 10^6 every 1 PVU
clevs_pv = np.arange(0, 25, 1)
cs1 = ax.contour(lons, lats, pv[i250]*1e6, clevs_pv, colors='black',
                 transform=datacrs)
plt.clabel(cs1, fmt='%d', fontsize='large')

# Plot the colorfill of divergence, scaled 10^5 every 1 s^1
clevs_div = np.arange(-15, 16, 1)
cs1 = ax.contourf(lons, lats, div[i250]*1e5, clevs_div, cmap=plt.cm.PuOr,
                  extend='both', transform=datacrs)
plt.colorbar(cs1, orientation='horizontal', pad=0, aspect=50, extendrect=True)

# Plot the wind barbs at 250 hPa
ax.barbs(lons, lats, uwnd[i250].to('kt').m, vwnd[i250].to('kt').m, pivot='middle',
         color='black', regrid_shape=20, transform=datacrs)

# Plot some titles to tell people what is on the map
plt.title('250-hPa GFS PV (PVU), Divergence (s$^{-1}$),'
          ' and Wind Barbs (kt)', loc='left')
plt.title('Valid Time: {}'.format(vtime), loc='right')

plt.savefig('250_hPa_PV_Analysis.png', dpi=150, bbox_inches='tight')
```

```
plt.close()
```

250-hPa GFS PV (PVU), Divergence (s^{-1}), and Wind Barbs (kt)

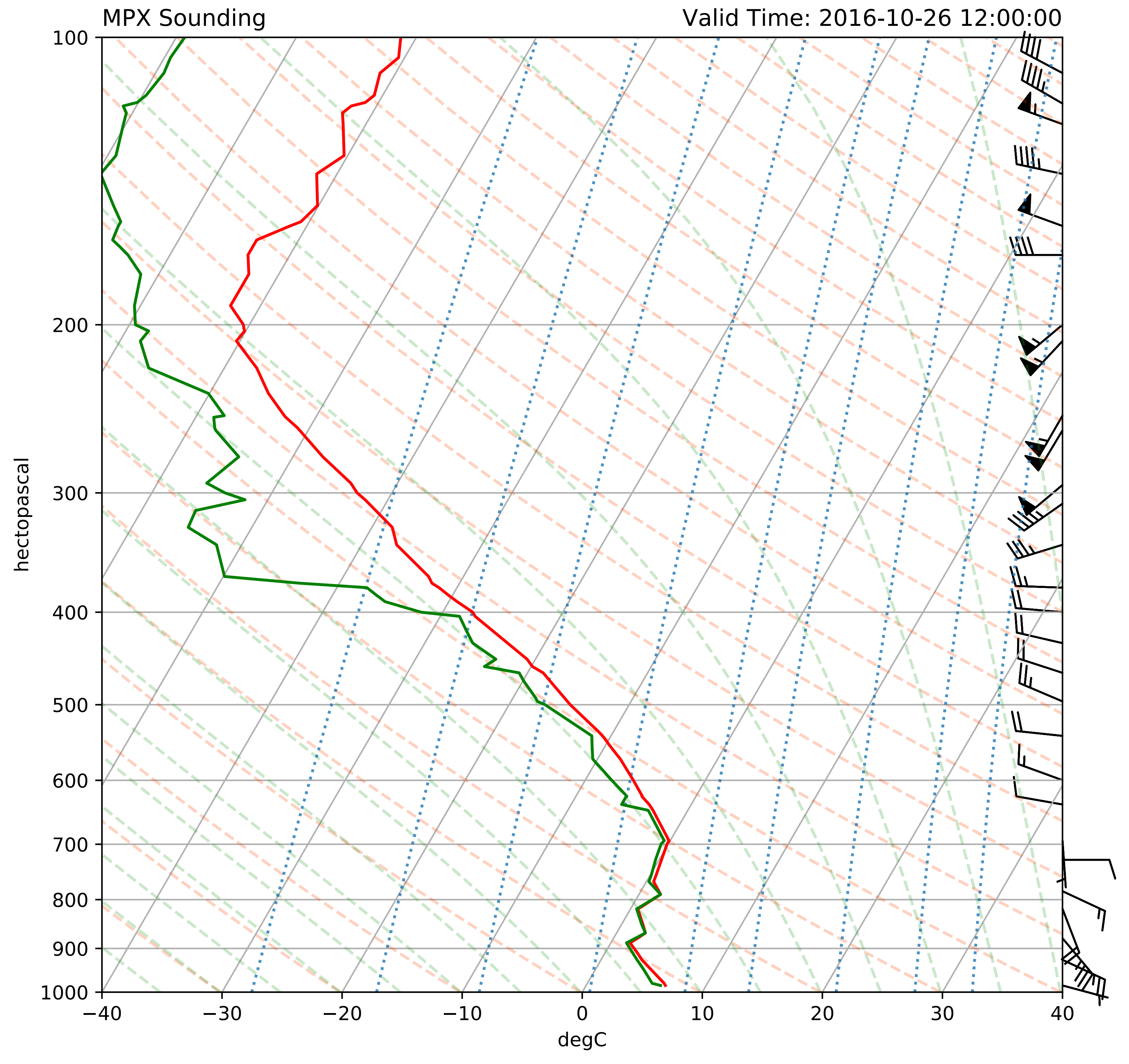
Valid Time: 2010-10-26 12:00:00



Skew-T Log-p Diagram Example

```python
"""
Skew-T Analysis
===============

Classic skew-T/log-p plot using data from University of Wyoming.

This example uses example data from the University of Wyoming sounding
archive for 12 UTC 31 October 2016 for Minneapolis, MN (MPX) and uses
MetPy to plot the classic skew-T with Temperature, Dewpoint, and wind
barbs.

"""

from datetime import datetime

import matplotlib.pyplot as plt
import metpy.calc as mpcalc
from metpy.plots import SkewT
from metpy.units import units
import numpy as np
from siphon.simplewebservice.wyoming import WyomingUpperAir


######################################################################
# Set time using a datetime object and station as variables
#

dt = datetime(2016, 10, 26, 12)
station = 'MPX'


######################################################################
# Grad Remote Data
# ----------------
#
# This requires an internet connection to access the sounding data from a
# remote server at the University of Wyoming.
#

df = WyomingUpperAir.request_data(dt, station)


######################################################################
# Isolate variables and attach units
#

p = df.pressure.values * units(df.units['pressure'])
T = df.temperature.values * units(df.units['temperature'])
Td = df.dewpoint.values * units(df.units['dewpoint'])
u = df.u_wind.values * units(df.units['u_wind'])
v = df.v_wind.values * units(df.units['v_wind'])

# Set index value for 100 hPa level
# This will help with plotting the final skew-T
# Defaults to last value in array if 100 hPa not in data
try:
    ip100 = list(p.m).index(100)+1
except:
    ip100 = -1


######################################################################
# Make Skew-T Plot
# ----------------
#
# The code below makes a basic skew-T plot using the MetPy plot module
# thaht contains a SkewT class.
#

# Change default to be better for skew-T
plt.rcParams['figure.figsize'] = (9, 9)

# Initiate the skew-T plot type from MetPy class loaded earlier
```

```
skew = SkewT()

# Plot the data using normal plotting functions, in this case using
# log scaling in Y, as dictated by the typical meteorological plot
skew.plot(p, T, 'r')
skew.plot(p, Td, 'g')
skew.plot_barbs(p[:ip100:3], u[:ip100:3], v[:ip100:3])

# Set some appropriate axes limits for x and y
skew.ax.set_xlim(-40, 40)
skew.ax.set_ylim(1000, 100)

# Add the relevant special lines to plot throughout the figure
skew.plot_dry_adiabats(t0=np.arange(233,533,10) * units.K,
                       alpha=0.25, color='orangered')
skew.plot_moist_adiabats(t0=np.arange(233,400,5) * units.K,
                         alpha=0.25, color='tab:green')
skew.plot_mixing_lines(p=np.arange(1000, 99, -20) * units.hPa,
                       linestyle='dotted', color='tab:blue')

# Add some descriptive titles
plt.title('{} Sounding'.format(station), loc='left')
plt.title('Valid Time: {}'.format(dt), loc='right')

plt.savefig('{0}_{1:%Y%m%d%H}_SkewT_Analysis.png'.format(station, dt),
            dpi=150, bbox_inches='tight')
plt.close()
```

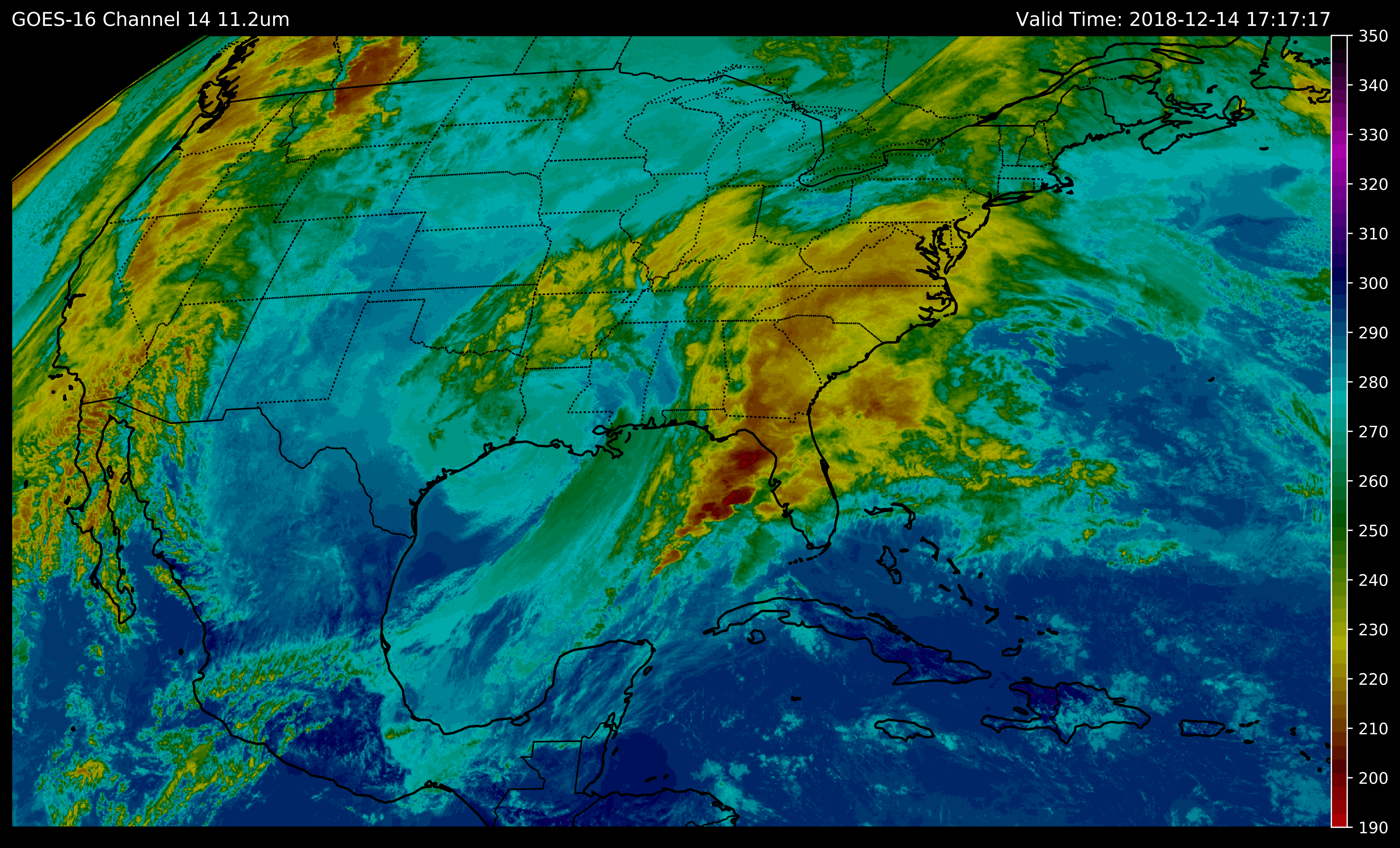


MPX Sounding
Valid Time: 2016-10-26 12:00:00
100
200
300
400
500
600
700
800
900
1000
hectopascal
−40
−30
−20
−10
0
10
20
30
40
degC

GOES 16 Infrared Satellite Imagery Example

```python
"""
Satellite Analysis
==================

Classic GOES 16 Infrared Satelite Analysis with colormap.

This example uses example data from the GOES 16 to plot the IR imagery
with a colortable for the most current data accessed using Siphon. The
plotting is completed with the help of MetPy and geographic plotting
using Cartopy for a CONUS view.

"""

from datetime import datetime

import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt
import metpy.calc as mpcalc
from metpy.plots import colortables
from metpy.units import units
import numpy as np
from scipy.ndimage import gaussian_filter
from siphon.catalog import TDSCatalog
import xarray as xr
plt.style.use('dark_background')


######################################################################
# Grab Data
# ---------
#
# The following code will access data from a THREDDS server at Unidata
# that contains GOES 16 data. Through siphon we can access the data using
# an xarray format.
#

# Set channel number (1-16)
# Channels 1-6 (Visible)
# Channel 7 (daylight/nighttime band)
# Channels 8-10 (Water Vapor)
# Channels 11-16 (IR)

# Have computer give current time
date = datetime.utcnow()

# Select a channel for viewing
channel = 14
sector = 'CONUS'

#https://thredds.ucar.edu/thredds/catalog/satellite/goes16/GOES16/CONUS/Channel14/cu
rrent/catalog.html
base_url = 'http://thredds.unidata.ucar.edu/thredds/catalog/satellite/goes16/GOES16/
'
date_portion = '{0}/Channel{1:02d}/{2:%Y%m%d}/'.format(sector, channel, date)
current_goes16 = TDSCatalog(base_url+date_portion+'catalog.xml')

# Use the latest file through the 0 index value
latest_file = current_goes16.datasets[0]

# Actually grab the data
ds = latest_file.remote_access(use_xarray=True)

# Use MetPy parsing to help gather projection information
sat_data = ds.metpy.parse_cf('Sectorized_CMI')

# Set projection based on info from file
proj = sat_data.metpy.cartopy_crs

# Get projection x and y data for plotting
x = sat_data.metpy.x
y = sat_data.metpy.y
```

```
# Have file tell me the central wavelength of the data
wavelength = ds.central_wavelength


######################################################################
# Plot Figure
# -----------
#
# The following code will plot the satellite image channel using a
# colormap from MetPy
#

# Start the figre
fig = plt.figure(figsize=(18, 9))
ax = fig.add_subplot(111, projection=proj)

# Use a colortable/colormap available from MetPy
ir_norm, ir_cmap = colortables.get_with_range('ir_drgb_r', 190, 350)

# Plot the data using imshow
im = ax.imshow(sat_data, origin='upper', cmap=ir_cmap, norm=ir_norm,
               extent=(x.min(), x.max(), y.min(), y.max()))
plt.colorbar(im, pad=0, aspect=50, ticks=range(190,351,10))

# Add country borders and states (use your favorite linestyle!)
ax.add_feature(cfeature.COASTLINE.with_scale('50m'), linewidth=1.5,
               edgecolor='black')
ax.add_feature(cfeature.BORDERS.with_scale('50m'), linewidth=1,
               edgecolor='black')
ax.add_feature(cfeature.STATES.with_scale('50m'), linestyle=':',
               edgecolor='black')

# Add appropriate titles
timestamp = datetime.strptime(ds.start_date_time, '%Y%j%H%M%S')
plt.title('Valid Time: {}'.format(timestamp), loc='right')
plt.title('GOES-16 Channel {} {:.1f}um'.format(channel, wavelength), loc='left')

plt.savefig('IR_Satellite.png', dpi=150, bbox_inches='tight')
plt.close()
```



GOES-16 Channel 14 11.2um
Valid Time: 2018-12-14 17:17:17
350
340
330
320
310
300
290
280
270
260
250
240
230
220
210
200
190