MetPy: Python Tools for Meteorological Data Analysis and Visualization

Ryan May

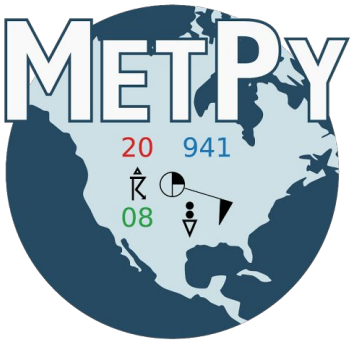
UCP/Unidata

10 June 2019

Unidata

- NSF-funded center for developing tools, data access, and community support
- Tools:
  - netCDF
  - LDM
  - IDV
  - THREDDS
  - Rosetta
  - AWIPS/GEMPAK
  - Python Training, **MetPy**, Siphon

What is MetPy?



Plots Calculations File I/O

- Toolkit for meteorological applications in Python
- Provide GEMPAK-like functionality in Python

Foundation

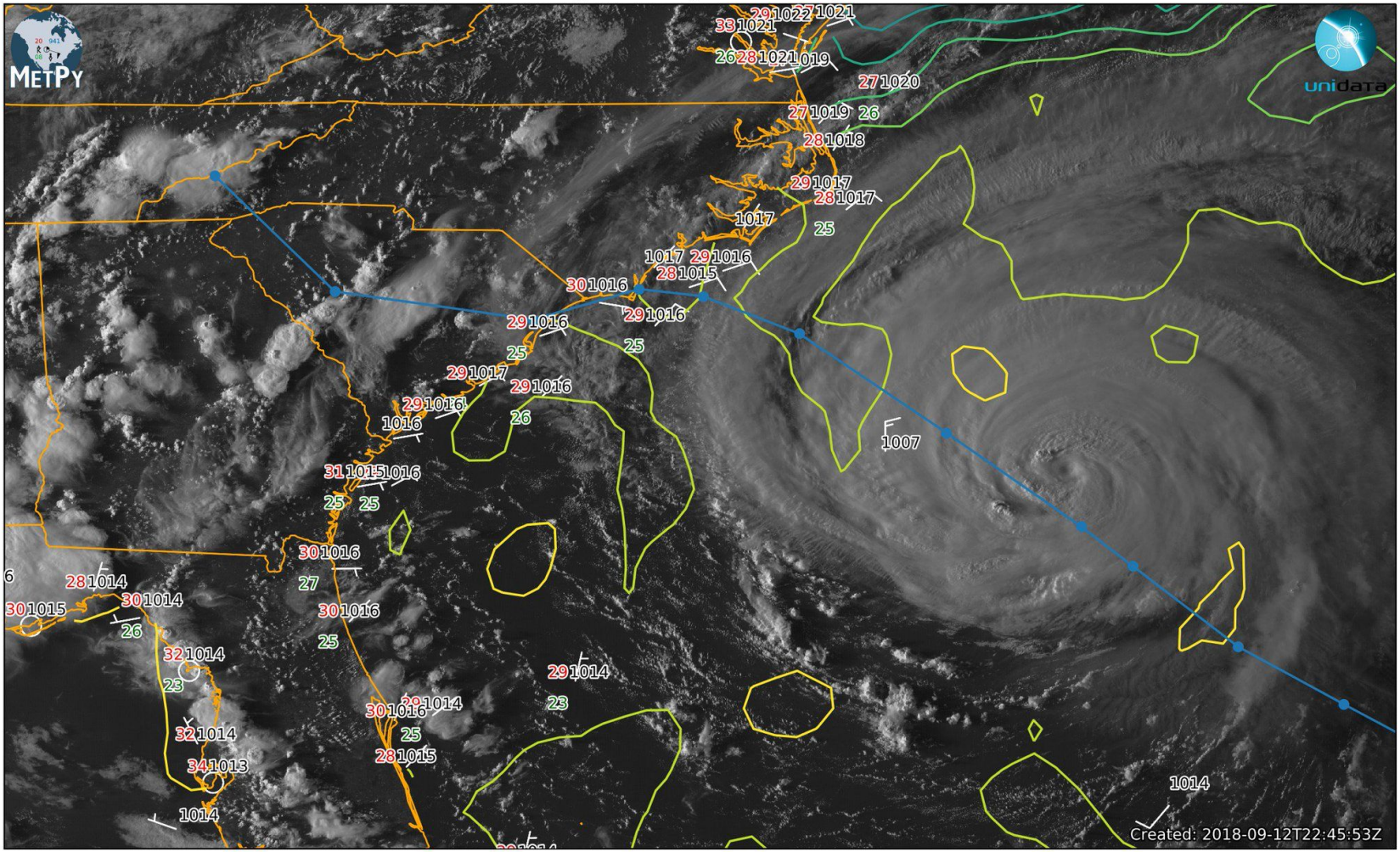
- Builds on many other Python tools:
  - NumPy
  - Matplotlib
  - SciPy
  - Pint
  - Xarray
  - Cartopy
- Test **everything**
- Automate as much as possible

Broad Goals

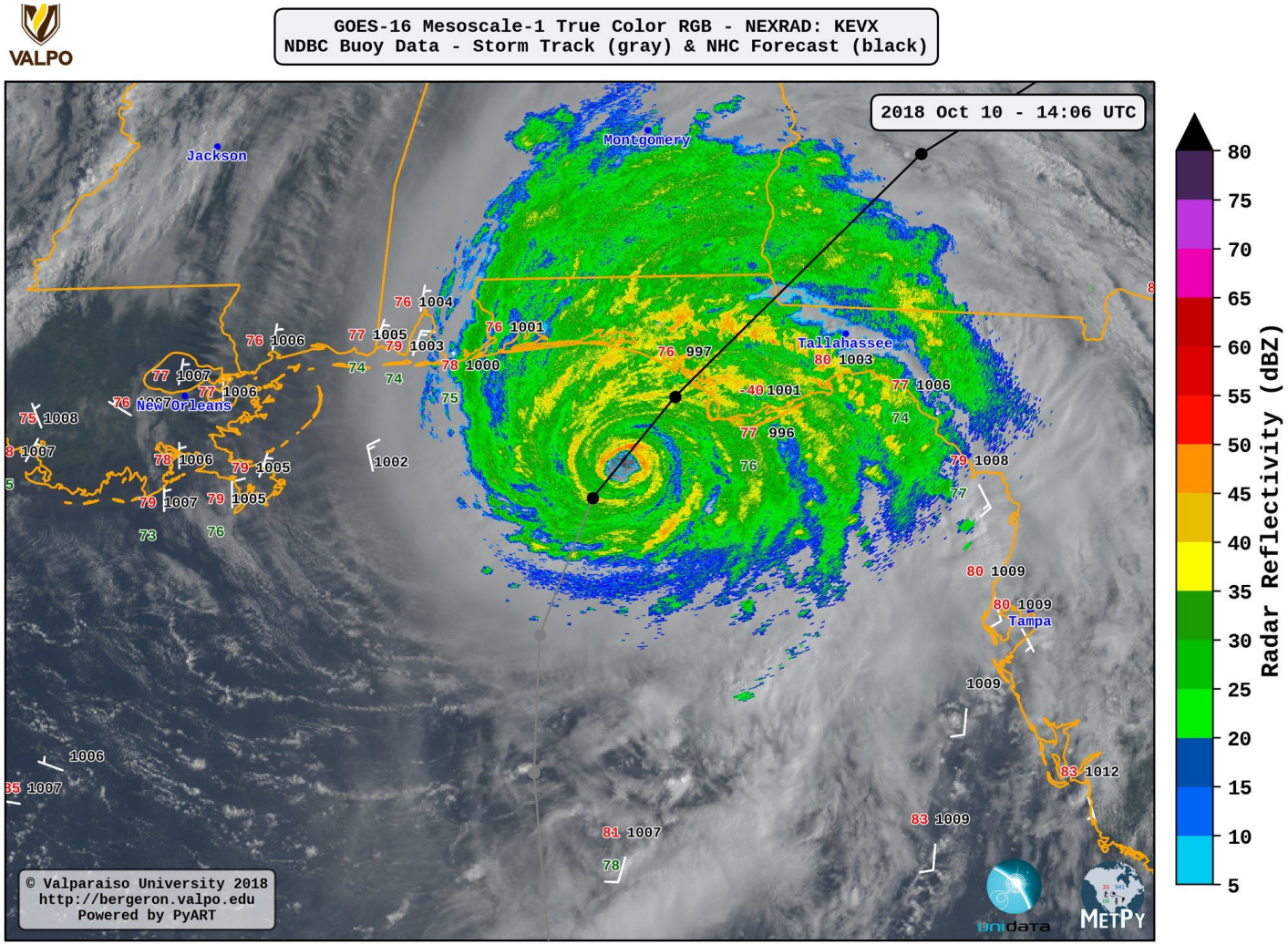
- Intended to be general purpose:
  - Education
  - Research
  - Other applications
- Be a community resource to find useful pieces
- Allow easily combining with other applications

Hurricane Florence

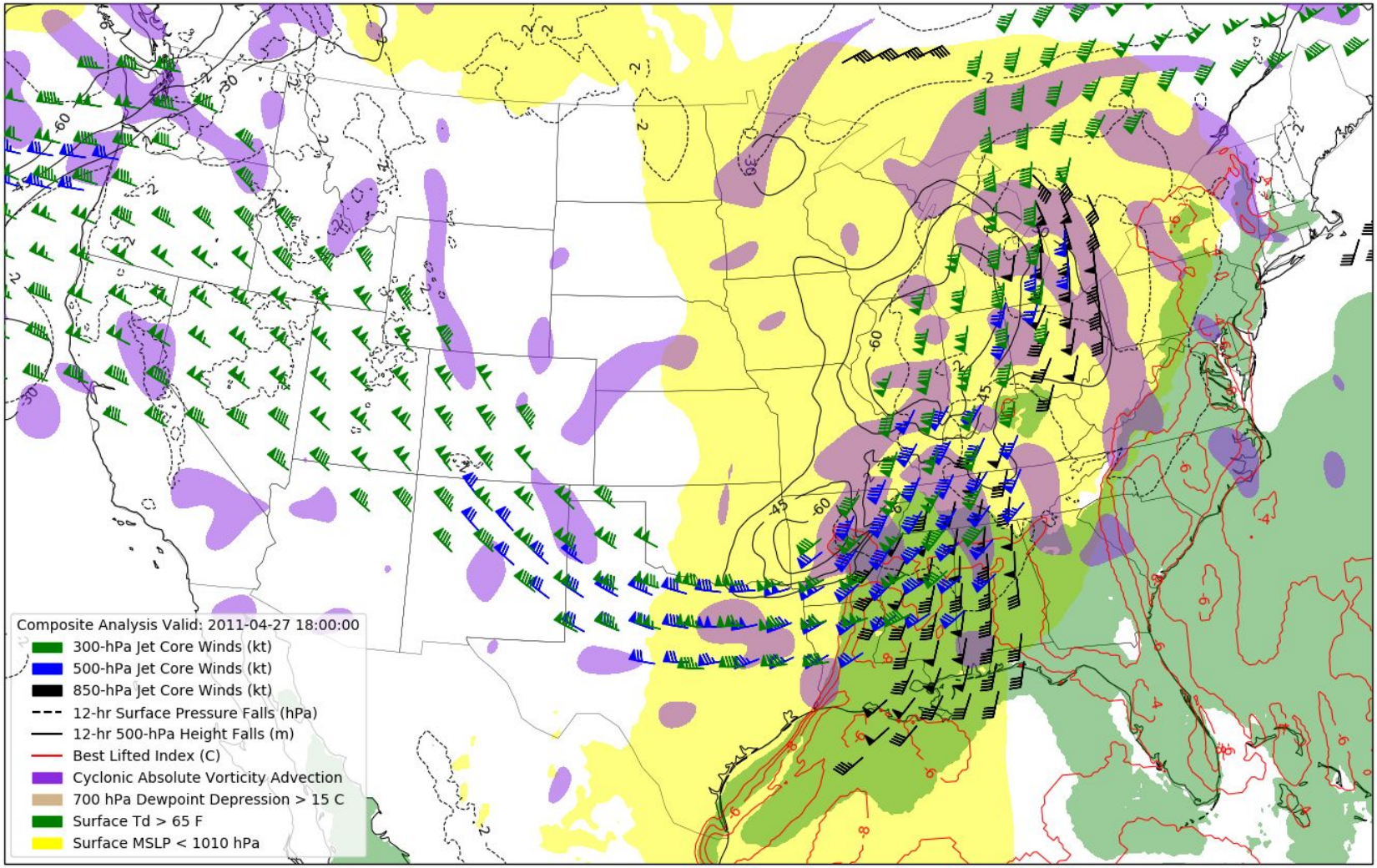
GOES-16 Visible, SST (contours), NDBC Buoy Observations, NHC Best and Forecast Track



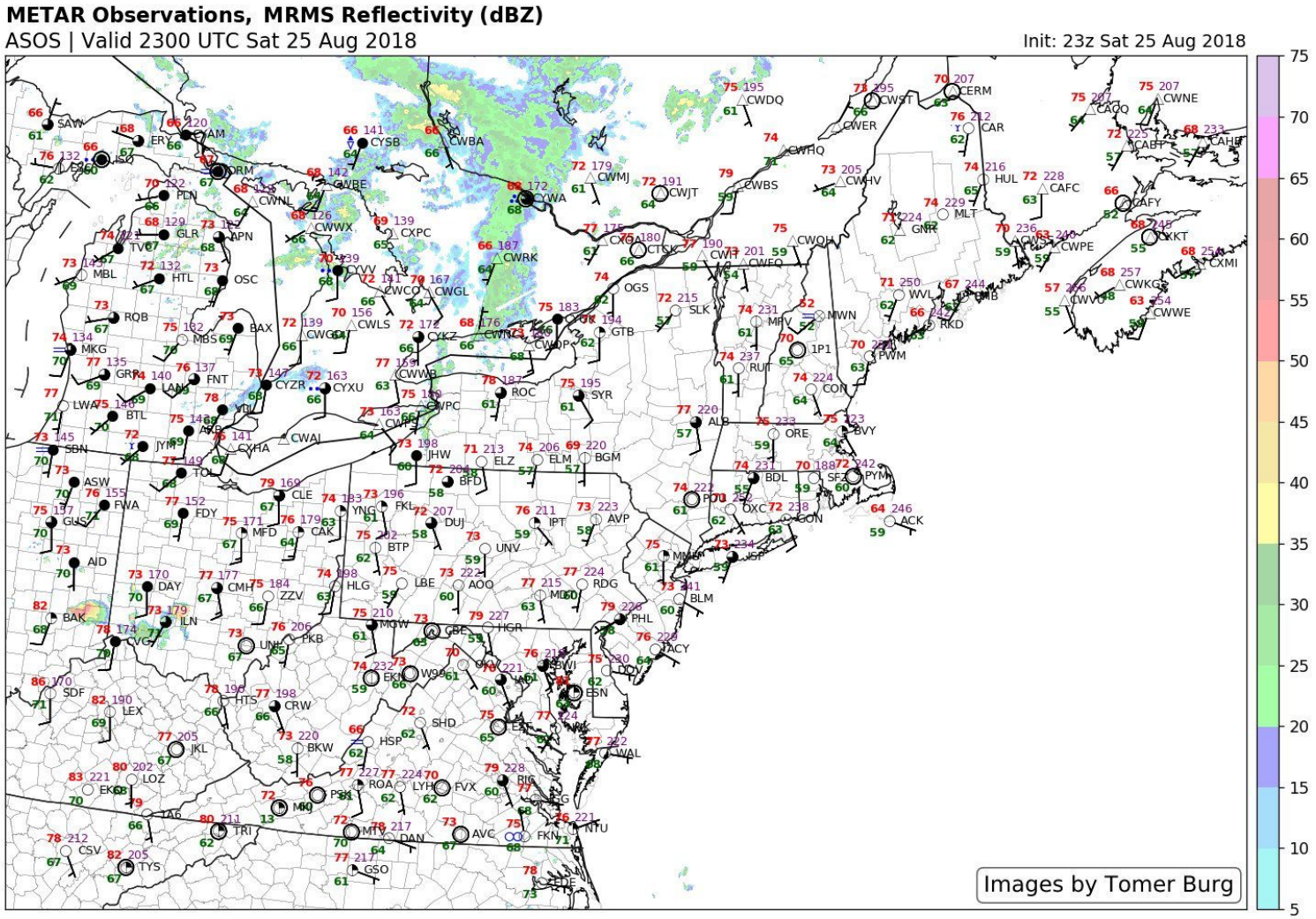
Hurricane Michael



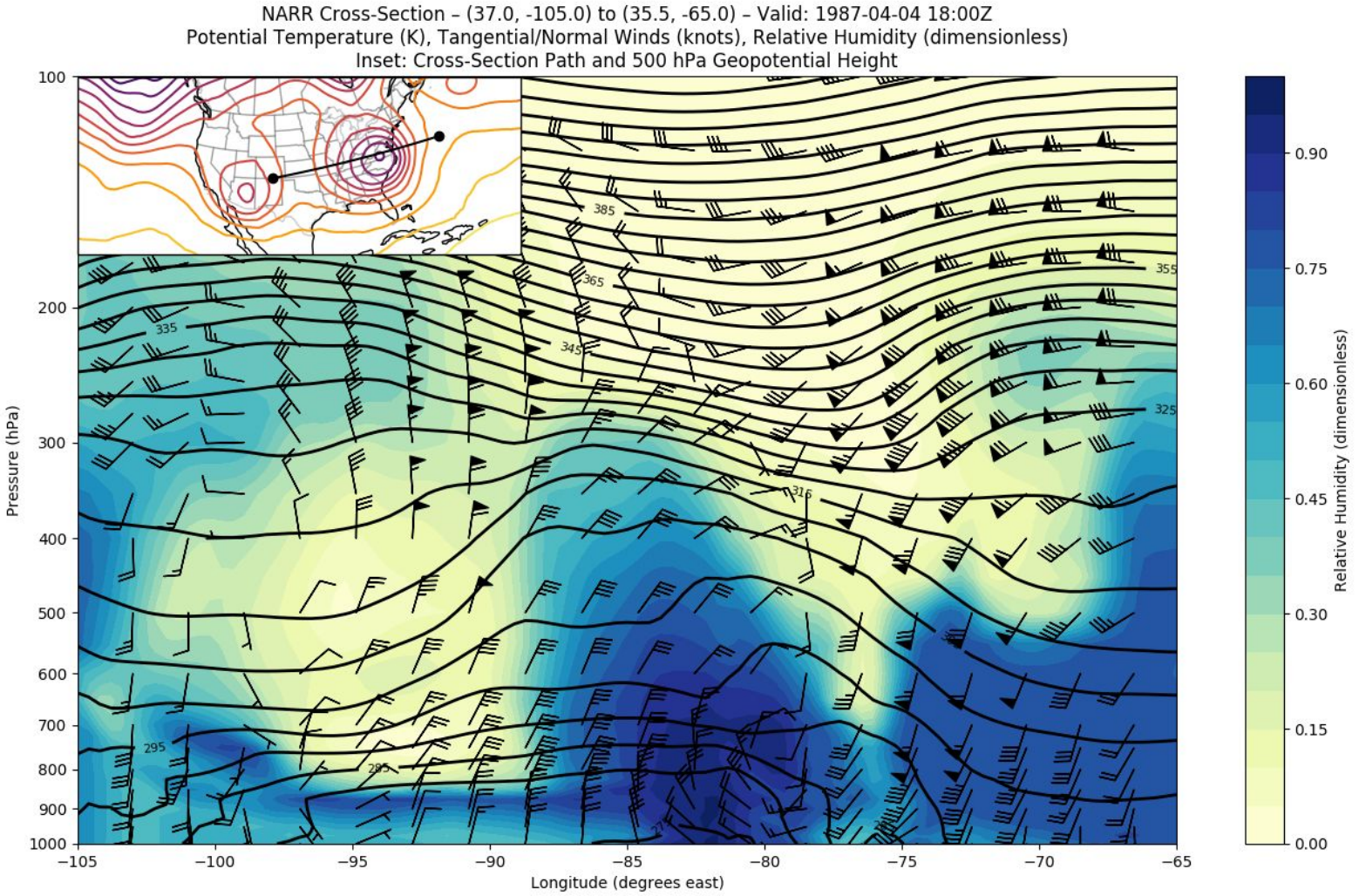
Miller Composite



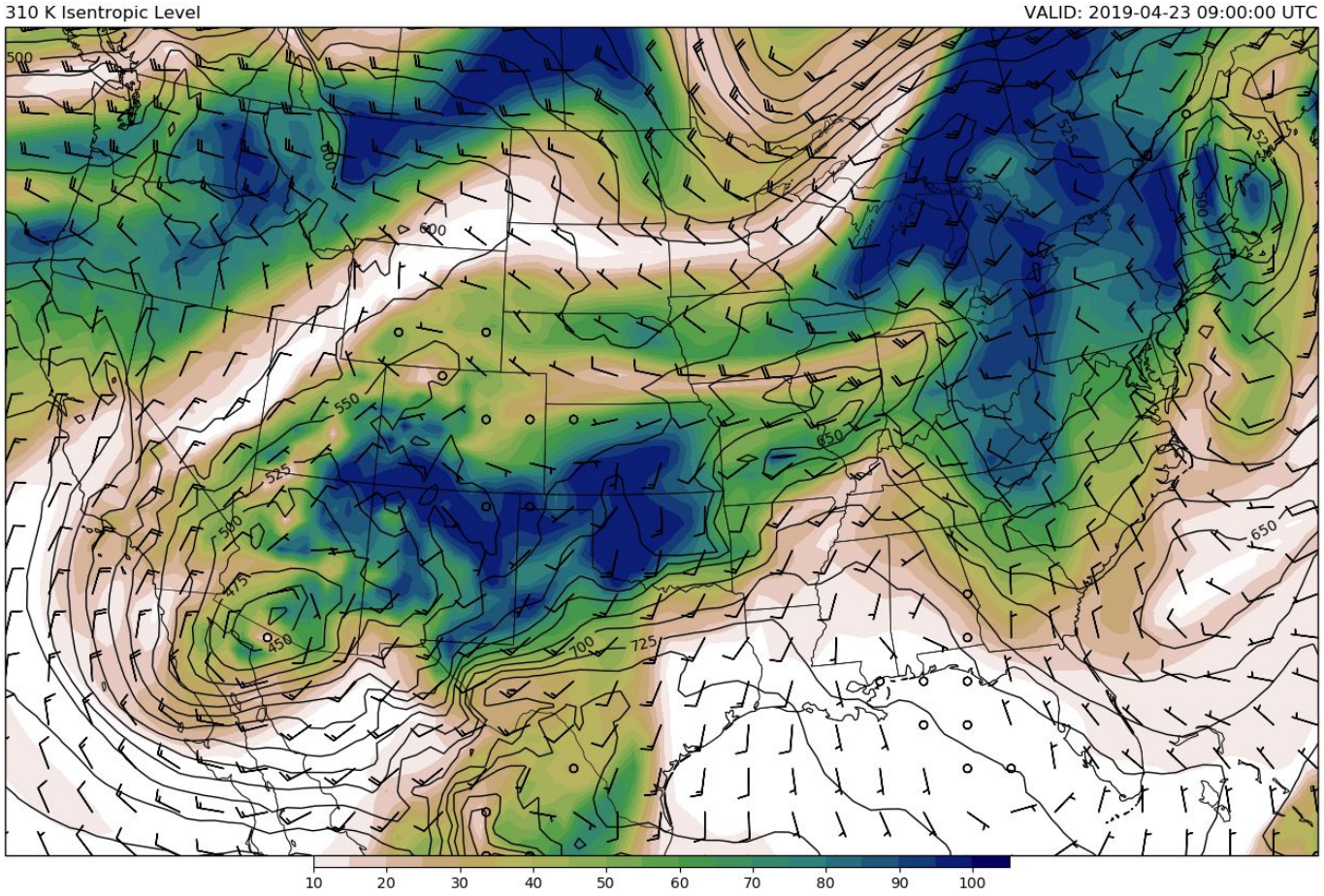
Station Model Plots



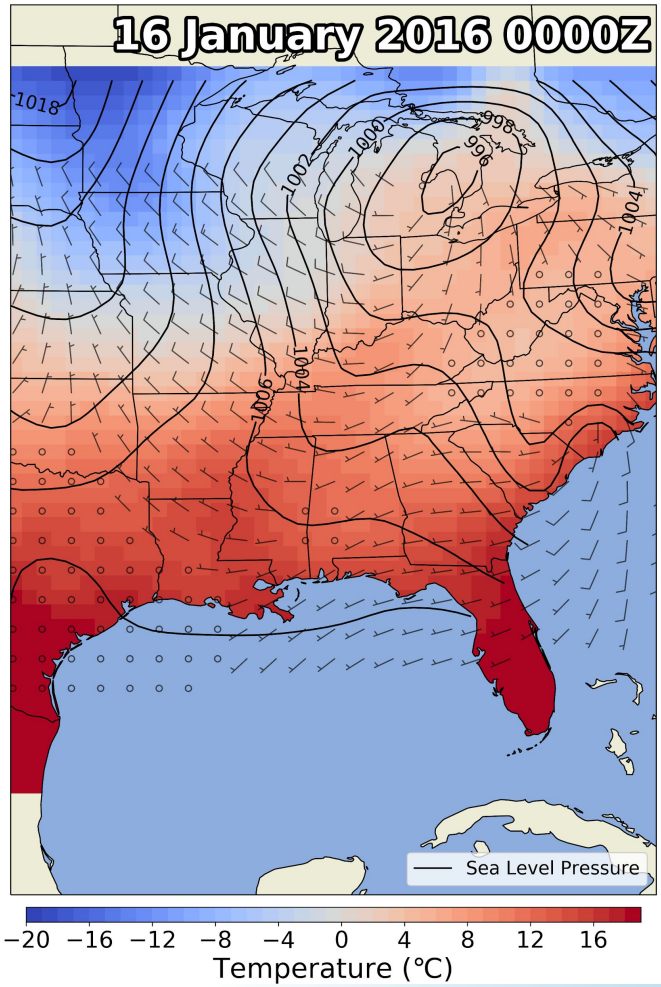
Cross-Sections



Isentropic Interpolation



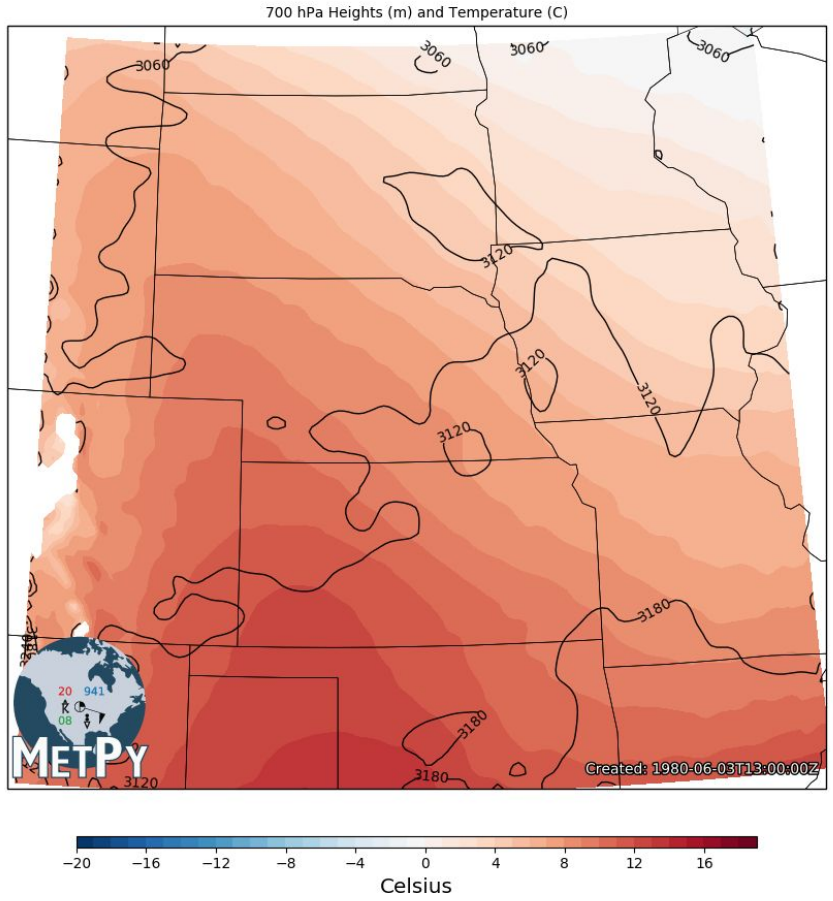
Gridding



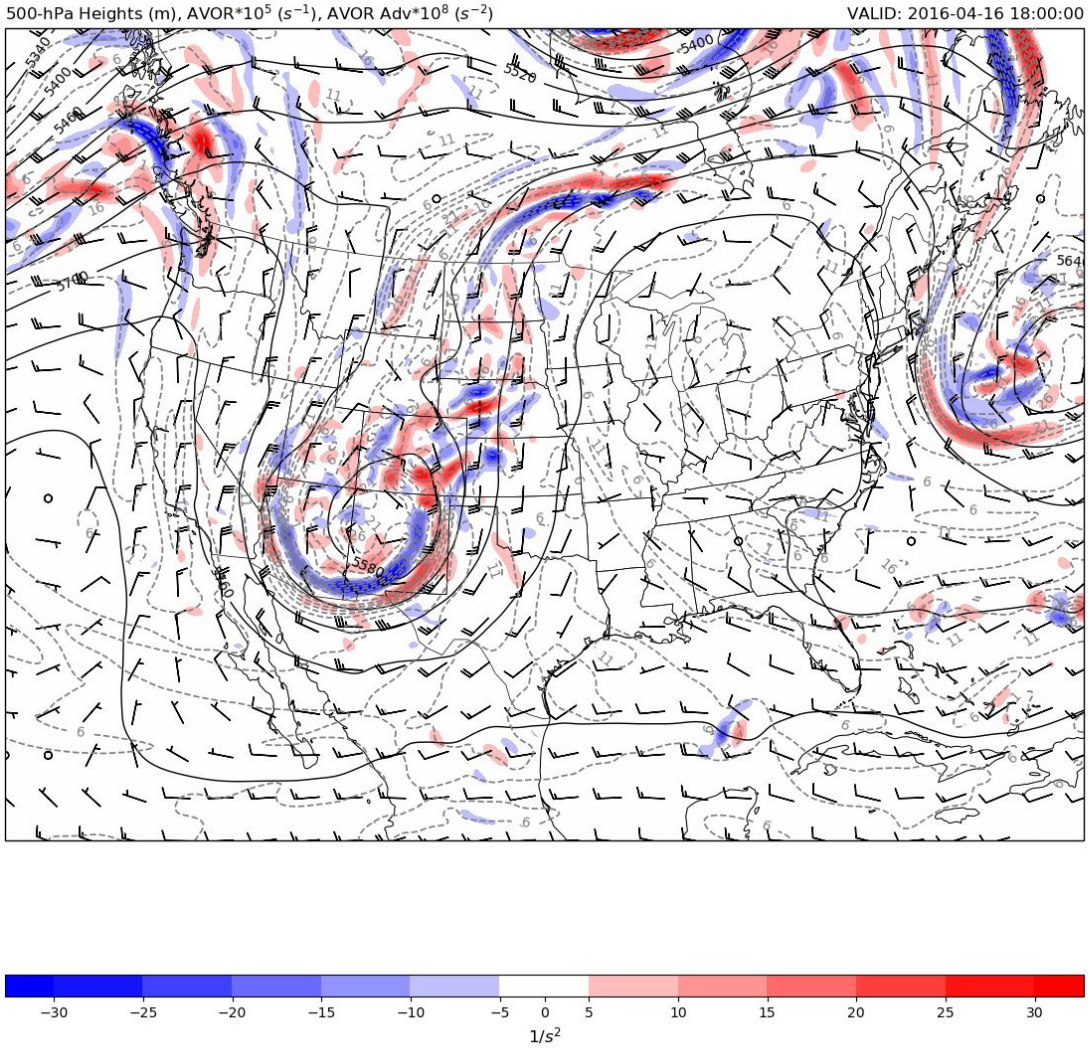
- Nearest Neighbor
- Barnes
- Cressman
- Linear
- Natural Neighbor

Interpolation & Derivatives

- Linear interpolation
- Log interpolation
- Derivative
- Gradient
- Laplacian



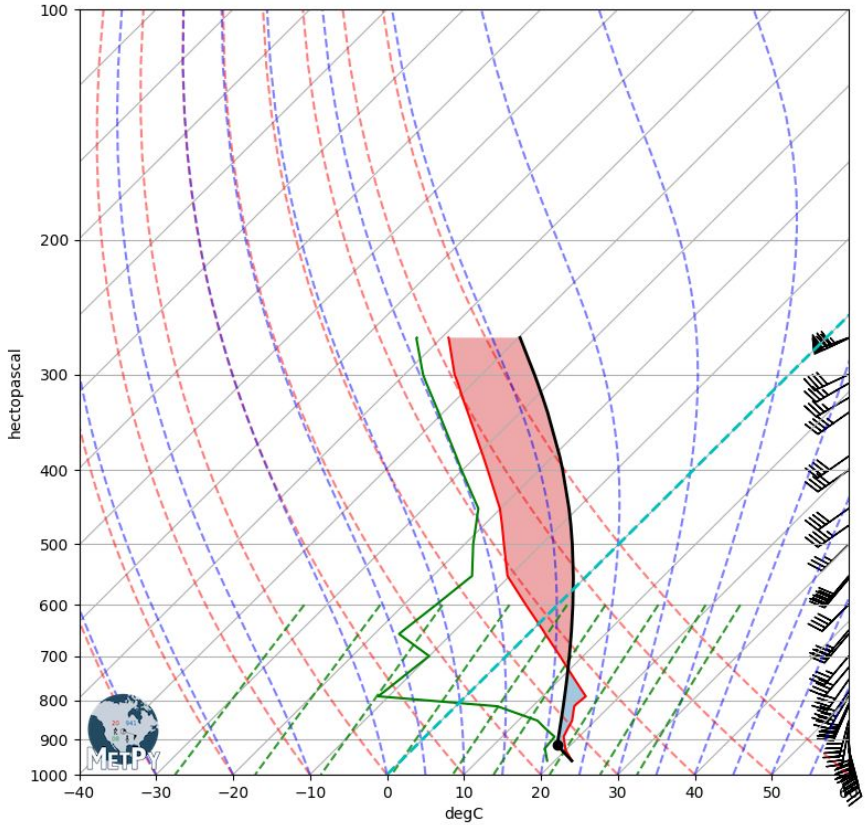
Kinematic Calculations



- Vorticity
- Divergence
- Deformation
- Frontogenesis
- Q-Vector
- Advection

Skew-T and Calculations

- Hodograph
- Sounding Calcs
  - CAPE/CIN
  - Storm Motion
  - LCL/LFC
  - Mixed Parcel
  - Supercell Composite



More Features

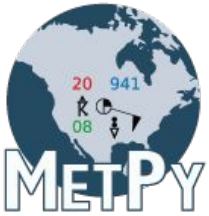
- Xarray integration and decoding CF-convention metadata
- Reading NEXRAD and GINI file formats
- US counties and matching state borders for Cartopy

Replacing GEMPAK

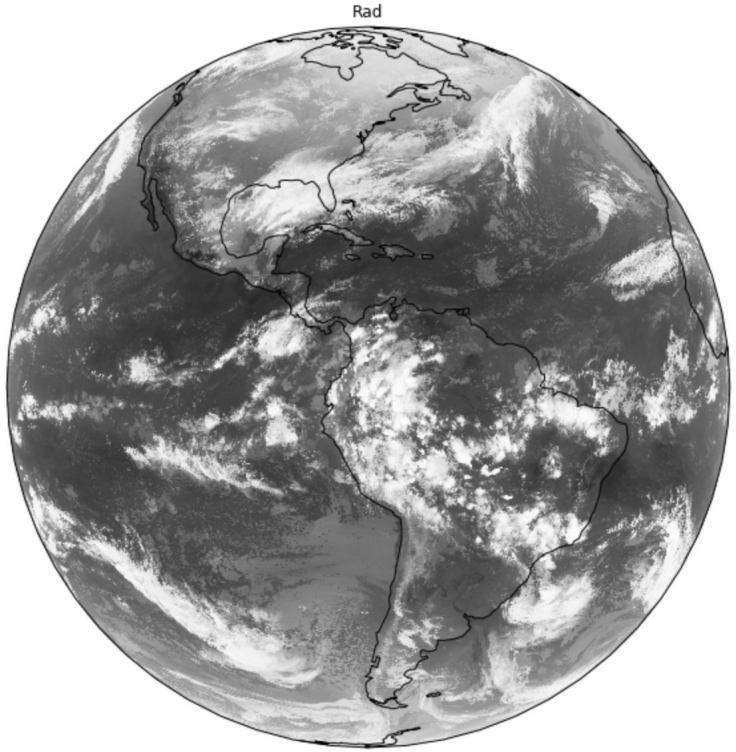
```
source /Users/gempak/GEMPAK6.3.0/Gemenviron
SET CURDAY = `date -u +%Y%m%d`
set FRUN = 12

gdcntr <<EOF1
  GDFILE    = gfs/${CURDAY}${FRUN}_gfs003.gem
  GDATTIM   = 'f012'
  GLEVEL    = 700
  GVCORD    = pres
  CTYPE     = f
  GFUNC     = avor(wnd)
  CONTUR    = 2
  CINT      = 2
  LINE      = 1/1
  TITLE     = 31/-2/GFS ~
  GAREA     = us
  PROJ      = 'str/90;-100;0'
  DEVICE    = 'gif|us.gif|1024;768'
  r


e
EOF1
```

```
from datetime import datetime

import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.gridspec as gridspec
import matplotlib.pylab as plt
import metpy.calc as mpcalc
from metpy.units import units
from netCDF4 import num2date
import numpy as np
import scipy.ndimage as ndimage
from siphon.ncss import NCSS

ncss = NCSS('https://www.ncei.noaa.gov/thredds/ncss/grid/namanl/'
            '201604/20160416/namanl_218_20160416_1800_000.grb')
now = datetime.utcnow()

hgt = ncss.query().time(datetime(2016, 4, 16, 18)).accept('netcdf')
hgt.variables('u-component_of_wind_isobaric',
              'v-component_of_wind_isobaric').add_lonlat()

ds = ncss.get_data(hgt)

lon = ds.variables['lon'][:]
lat = ds.variables['lat'][:]

times = ds.variables[ds.variables['u-component_of_wind_isobaric'].dimensions[0]]
vtime = num2date(times[:], units=times.units)

lev_500 = np.where(ds.variables['isobaric'][:] == 500)[0][0]

uwnd_500 = units('m/s') * ds.variables['u-component_of_wind_isobaric'][0, lev_500, :, :]
vwnd_500 = units('m/s') * ds.variables['v-component_of_wind_isobaric'][0, lev_500, :, :]

dx, dy = mpcalc.lat_lon_grid_deltas(lon, lat)

avor = mpcalc.absolute_vorticity(uwnd_500, vwnd_500, dx, dy,
                                 lat * units.degrees, dim_order='yx')

avor = ndimage.gaussian_filter(avor, sigma=3, order=0) * units('1/s')

dproj = ds.variables['LambertConformal_Projection']
globe = ccrs.Globe(ellipse='sphere', semimajor_axis=dproj.earth_radius,
                   semiminor_axis=dproj.earth_radius)
datacrs = ccrs.LambertConformal(central_latitude=dproj.latitude_of_projection_origin,
                                central_longitude=dproj.longitude_of_central_meridian,
                                standard_parallels=[dproj.standard_parallel],
                                globe=globe)
plotcrs = ccrs.LambertConformal(central_latitude=45., central_longitude=-100.,
                                standard_parallels=[30, 60])

fig = plt.figure(1, figsize=(14., 12))
gs = gridspec.GridSpec(2, 1, height_ratios=[1, .02], bottom=.07, top=.99,
                       hspace=0.01, wspace=0.01)
ax = plt.subplot(gs[0], projection=plotcrs)

plt.title(r'AVOR$*10^5$ ($s^{-1}$)', loc='left')
plt.title('VALID: {}'.format(vtime[0]), loc='right')

ax.set_extent([235., 290., 20., 58.], ccrs.PlateCarree())
ax.coastlines('50m', edgecolor='black', linewidth=0.75)
ax.add_feature(cfeature.STATES, linewidth=.5)

clevvort500 = np.arange(-9, 50, 5)
cs2 = ax.contour(lon, lat, avor*10**5, clevvort500, colors='grey',
                 linewidths=1.25, linestyles='dashed', transform=ccrs.PlateCarree())
plt.clabel(cs2, fontsize=10, inline=1, inline_spacing=10, fmt='%i',
           rightside_up=True, use_clabeltext=True)

gs.tight_layout(fig)
plt.show()
```

MetPy Simplified Plotting




```
grb_cat = TDSCatalog('http://thredds.ucar.edu/thredds/'
                     'catalog/satellite/goes16/GRB16/ABI/'
                     'FullDisk/Channel11/current/catalog.xml')
grb_dat = grb_cat.datasets[0].remote_access(use_xarray=True)

img = ImagePlot()
img.data = grb_dat
img.field = 'Rad'
img.colormap = 'Greys'

m = MapPanel()
m.projection = 'data'
m.plots = [img]

c = PanelContainer()
c.size = (10, 10)
c.panels = [m]
c.draw()
```



Design Philosophy

- Fit well with scientific Python ecosystem
- Simple to use with your own data
- Unit-correctness built-in (using pint)
- Good online documentation

Documentation Sample

virtual_temperature

metpy.calc.virtual_temperature(*temperature, mixing, molecular_weight_ratio=<Quantity(0.6219800858985514, 'dimensionless')>*) [source]

Calculate virtual temperature.

This calculation must be given an air parcel's temperature and mixing ratio. The implementation uses the formula outlined in [Hobbs2006] pg.80.

Parameters:

- **temperature** (*pint.Quantity*) – The temperature
- **mixing** (*pint.Quantity*) – dimensionless mass mixing ratio
- **molecular_weight_ratio** (*pint.Quantity* or float, optional) – The ratio of the molecular weight of the constituent gas to that assumed for air. Defaults to the ratio for water vapor to dry air. ($\epsilon \approx 0.622$).

Returns: *pint.Quantity* – The corresponding virtual temperature of the parcel

Notes

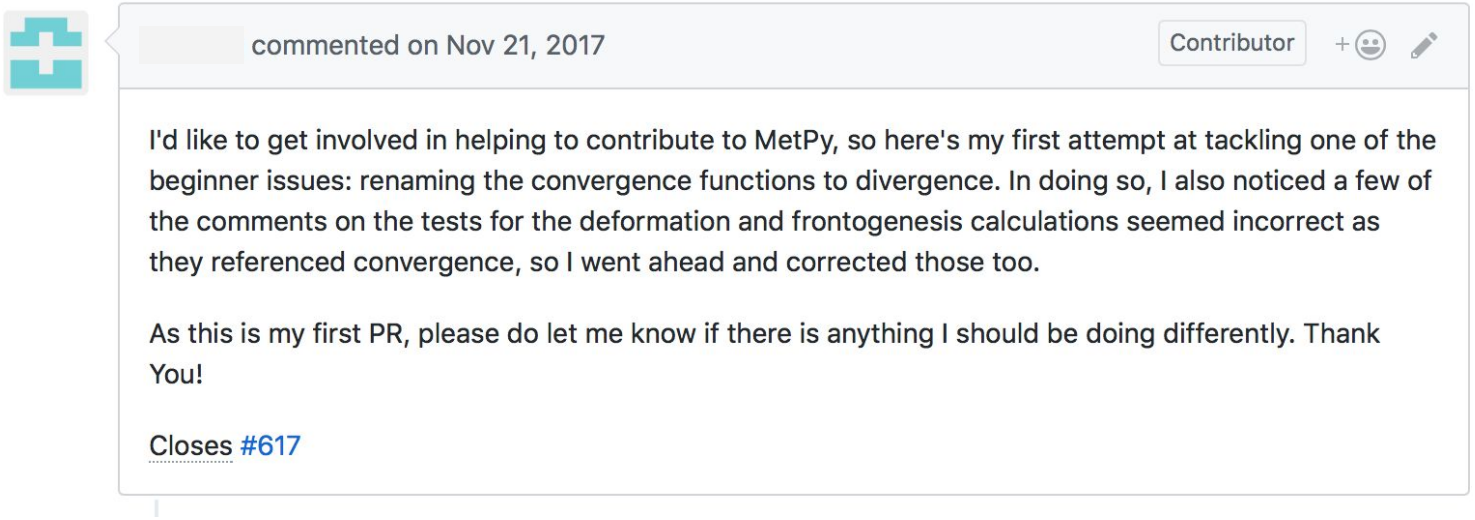
$$T_v = T\frac{\mathrm{w} + \epsilon}{\epsilon\,(1 + \mathrm{w})}$$

Community-Driven Development

- Actively soliciting community contributions and involvement
- Open development on GitHub
- Roadmap available in the web documentation
  - We welcome input on what to add!

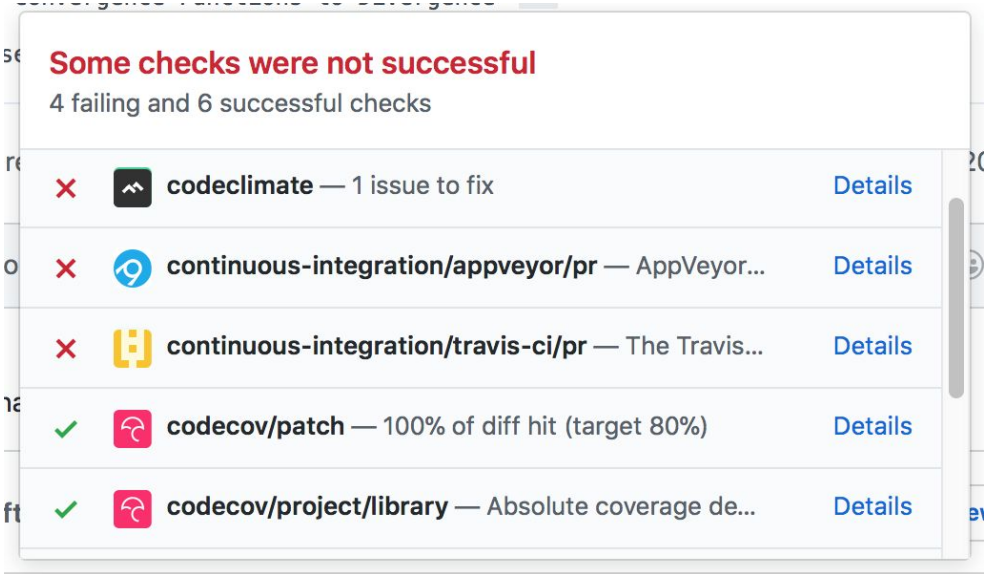
How can you contribute?

- Write code, docs, or an example
  - See our “Good First Issues” for inspiration
- Open a Pull Request



Contributing (cont.)

- Tests run automatically
- Get feedback



dopplershift requested changes on Nov 21, 2017

View changes

Welcome, and thanks for the contribution.

Overall this looks great and gets us closer to the 0.7.0 milestone, so thanks for the help. Just a few minor things to fix up and this will be ready.

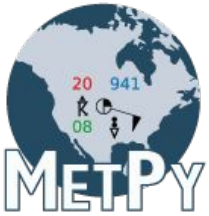
Contributing (cont.)

- It gets merged!

Everything Open

- All contributions go through this process
- Feel free to ask questions or comment on issues
- Or chat on Gitter
- Also can ask questions using the “metpy” tag on StackOverflow

Coming Attractions



- MetPy 1.0 release anticipated Fall 2019
  - **DROPPING Python 2.7 support**
- Expand simplified plotting interface
- More calculations (e.g. dynamic tropopause)
- More file format support (e.g. METAR, BUFR)
- See roadmap for more information

Resources

- GitHub: https://github.com/Unidata/MetPy
- Documentation: https://unidata.github.io/MetPy
- Follow @metpy on Twitter