wrf_python_instructor

June 23, 2017

1 WRF-Python Tutorial 2017

1.1 Bill Ladwig

1.2 NCAR/CISL/VAST

2 Topics

1. Introduction to jupyter notebooks, numpy, xarray
2. Overview of WRF-ARW Output Data
3. wrf-python
4. Plotting
5. Advanced

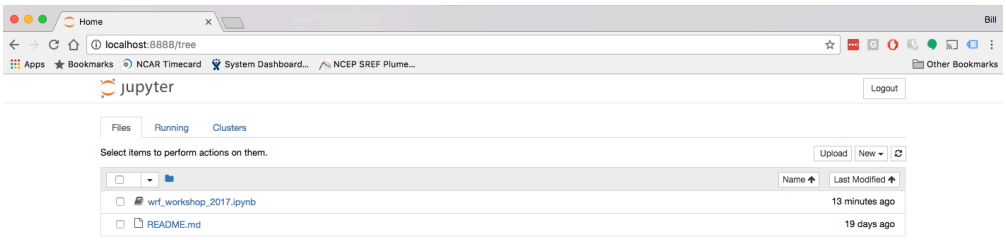
3 1.0 Introduction to jupyter, numpy, xarray

3.1 What is Jupyter Notebook?

- Originally IPython Notebook (now Julia, Python, R)
- The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text.
- The Jupyter Notebook actually consists of a document and an application
- The Jupyter Notebook application is a web browser application that allows editing and executing of jupyter notebook documents.

3.2 What is Jupyter Notebook?

- Jupyter Notebook documents (usually ending with a .ipynb extension), are really just JSON-formatted text files that contain the code and rich text elements that will be rendered by the jupyter notebook application.
- Jupyter notebook documents are NOT Python scripts, so do not try to run them via the 'python' command. They need to be converted first.
- For this tutorial, when we refer to jupyter notebook, we're referring to both the application and document.



alt

3.3 Activating Your Conda Environment

If you followed the installation instructions on the web, you should have created a "tutorial_2017" conda environment.

To activate it, first open a terminal and type in:

```
source activate tutorial_2017 [Linux/Mac]

activate tutorial_2017 [Windows]
```

Raise your hand if you need help.

3.4 Pulling Down the Latest Changes

```
cd ~/wrf_python_tutorial

[cd %HOMEPATH%\wrf_python_tutorial]

git checkout -- wrf_workshop_2017.ipynb

git pull
```

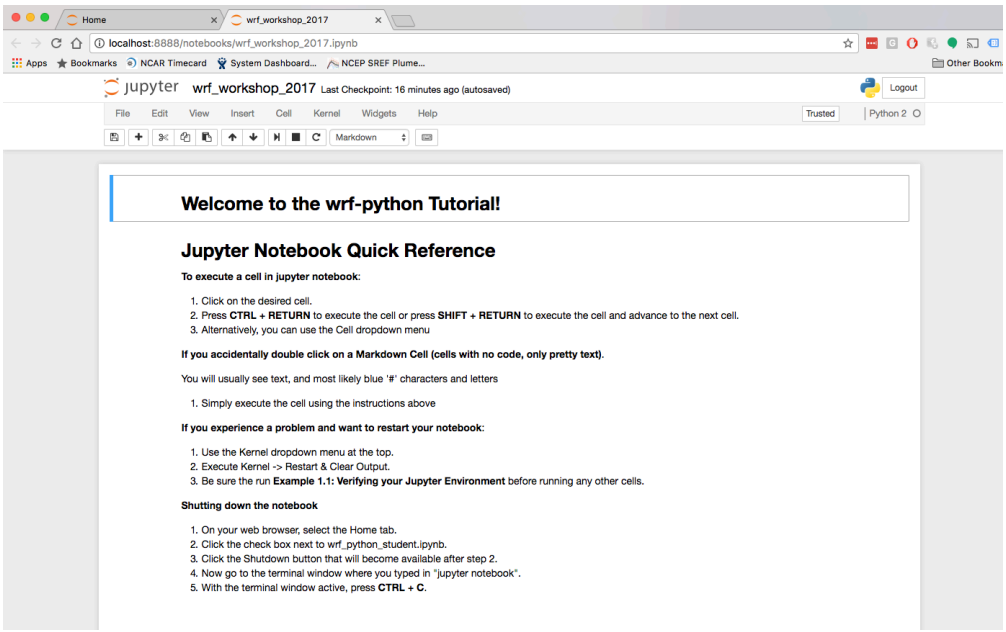
3.5 Starting jupyter notebook

```
cd ~/wrf_python_tutorial

[cd %HOMEPATH%\wrf_python_tutorial]

jupyter notebook
```

If for some reason the browser does not launch automatically (Mac Sierra Users), open a web browser and copy the URL listed on your command terminal:

```
http://localhost:8888/?token=...
```

Your web browser should look similar to this:
Now click on the **wrf_python_student.ipynb** link.
This should open a new browser tab that looks like:



alt

4 Cells

- A jupyter notebook is a collection of cells, similar to Mathematica.
- Cells can be either executable code or text (markdown).
- Cells can also be specified as slides, which is how this slide show was made (along with the Rise plugin).

5 Cells

- Entering and executing code in cells is the same as having typed it in to the Python shell program.
- The order of execution of the cells can have impacts on variables that are used across the cells, so be careful when re-running cells.
- Aside from the first cell you run, the cells used in this tutorial should be more like independent scripts.

5.1 Executing Cells

1. Click on the desired cell.
2. Press **CTRL + RETURN** to execute the cell or press **SHIFT + RETURN** to execute the cell and advance to the next cell.
3. Alternatively, you can use the Cell dropdown menu

5.2 Restarting the Notebook

If your notebook crashes for some reason:

1. Use the Kernel dropdown menu at the top.
2. Execute Kernel -> Restart & Clear Output.

5.3 Shutting down the notebook

1. On your web browser, select the Home tab.
2. Click the check box next to wrf_python_student.ipynb.
3. Click the Shutdown button that will become available after step 2.
4. Now go to the terminal window where you typed in "jupyter notebook".
5. With the terminal window active, press **CTRL + C**.

6 1.1 Verifying your Jupyter Environment

To set up the tutorial to work with your files, **modify the WRF_DIRECTORY and WRF_FILES variables** to point to your WRF files.

IMPORTANT: If for some reason your workbook crashes, you need to run this cell again before running the later examples.

In [1]:
```
from __future__ import print_function

# This jupyter notebook command inserts matplotlib graphics in
# to the workbook
%matplotlib inline

# Modify these to point to your own files
WRF_DIRECTORY = "~/wrf_tutorial_data"
WRF_FILES = ["wrfout_d01_2005-08-28_00_00_00",
             "wrfout_d01_2005-08-28_12_00_00",
             "wrfout_d01_2005-08-29_00_00_00"]


# Do not modify the code below this line
#-------------------------------------------------------
# Turn off annoying warnings
import warnings
warnings.filterwarnings('ignore')

# Make sure the environment is good
import numpy
import cartopy
import matplotlib
from netCDF4 import Dataset
from xarray import DataArray
from wrf import (getvar, interplevel, vertcross,
                 vinterp, ALL_TIMES)
import os

_WRF_FILES = [os.path.abspath(os.path.expanduser(
```

```
        os.path.join(WRF_DIRECTORY, f))) for f in WRF_FILES]

    # Check that the WRF files exist
    for f in _WRF_FILES:
        if not os.path.exists(f):
            raise ValueError("{} does not exist. "
                "Check for typos or incorrect directory.".format(full_path))

    # Create functions so that the WRF files only need
    # to be specified using the WRF_FILES global above
    def single_wrf_file():
        global _WRF_FILES
        return _WRF_FILES[0]

    def multiple_wrf_files():
        global _WRF_FILES
        return _WRF_FILES

    print ("All tests passed!")
```

```
All tests passed!
```

7 Numpy

- Numpy is a Python package for performing array based operations, similar to Matlab and NCL.
- Numpy arrays can be created for the common types in C (named "dtype" in numpy)
- int8, int16, int32, int64 (and unsigned versions)
- float16, float32, float64 [default]
- bool
- complex64, complex128
- Arrays can be C-ordered (fastest on right) or Fortran-ordered (fastest on left). C-ordered by default.

8 Numpy Basics

8.1 Array Creation

In this example, we're going to create an array of all zeros with 3x3x3 shape.

Here is how to create an array of floats and then integers.

```
import numpy

array_float32 = numpy.zeros((3,3,3),
```

```
                                "float32")

array_int32 = numpy.zeros((3,3,3), "int32")
```

8.2 Accessing Elements

- To access elements in numpy, you use the bracket "[ ]" syntax.
- Supply each desired index separated by commas (this is really a tuple).
- You can use also negative indexes to pick indexes from the end.

```
import numpy

my_array = numpy.zeros((3,3,3), "float32")

# Getting elements
first_element = my_array[0,0,0]

last_element = my_array[-1,-1,-1]

mid_element = my_array[1,1,1]

# Setting an element
my_array[1,1,1] = 10.0
```

8.3 Slices

- Slices are a way to extract array subsets from an array.
- The syntax for a slice is the ':' character.
- Specifying the ':' for a dimension will return all values along that dimension.
- Specifying 'start : end' will take a subset of that dimension. Also, either *start* or *end* can be left blank.
- Can also use a *step* value with 'start : end : step' if you want to increment values with something other than 1.

```
import numpy

my_array = numpy.zeros((3,3,3), "float32")

first_row = my_array[0,0,:]

first_column = my_array[0,:,0]

first_z = my_array[:,0,0]

subset = my_array[:, :, 1:3]

reverse_z = my_array[::-1, :, :]
```

Also, slices are implicitly applied from left to right for unspecified dimensions.

```
import numpy

my_array = numpy.zeros((3,3,3), "float32")

first_plane = my_array[0,:,:]

# This is the same as first_plane
first_plane2 = my_array[0]

# A short way to get everything
# Same as my_array[:,:,:]
all_elements = my_array[:]
```

9 Masked Arrays

- numpy uses a numpy array subclass called a MaskedArray.
- MaskedArrays contain a data array and a mask array.
- Usually a fill_value is set in the data array at each location where the mask array is True.
- Numerous ways to convert a regular numpy array to a MaskedArray:
- masked_equal
- masked_greater
- masked_where

9.1 Creating a MaskedArray

```
import numpy
import numpy.ma

my_array = numpy.zeros((3,3,3), "float32")

# Now all the array elements are masked values
my_masked = numpy.ma.masked_equal(my_array, 0)
```

10 Your Turn!

10.1 Example 1.2: Numpy Basics

```
In [2]: import numpy
        import numpy.ma

        my_array = numpy.zeros((3,3,3), "float32")

        print ("my_array")
        print (my_array)
```

```
        print ("\n")

        # Setting an element
        my_array[1,1,1] = 10.0

        # Getting an element
        mid = my_array[1,1,1]

        print ("Mid element set")
        print (my_array)
        print ("\n")

        # Getting a slice
        my_slice = my_array[1,:,:]

        print ("my_slice")
        print (my_slice)
        print ("\n")

        # Masking the zeros
        my_masked = numpy.ma.masked_equal(my_array, 0)

        print ("my_masked")
        print (my_masked)
        print ("\n")
```

```
my_array
[[[ 0.  0.  0.]
  [ 0.  0.  0.]
  [ 0.  0.  0.]]

 [[ 0.  0.  0.]
  [ 0.  0.  0.]
  [ 0.  0.  0.]]

 [[ 0.  0.  0.]
  [ 0.  0.  0.]
  [ 0.  0.  0.]]]


Mid element set
[[[  0.   0.   0.]
  [  0.   0.   0.]
  [  0.   0.   0.]]

 [[  0.   0.   0.]
  [  0.  10.   0.]
  [  0.   0.   0.]]
```

```
 [[  0.   0.   0.]
  [  0.   0.   0.]
  [  0.   0.   0.]]]


my_slice
[[  0.   0.   0.]
 [  0.  10.   0.]
 [  0.   0.   0.]]


my_masked
[[[-- -- --]
  [-- -- --]
  [-- -- --]]

 [[-- -- --]
  [-- 10.0 --]
  [-- -- --]]

 [[-- -- --]
  [-- -- --]
  [-- -- --]]]
```

11 xarray

- xarray expands upon numpy by adding dimension names, coordinate variables, and metadata.

- xarray array (DataArray) objects wrap around a numpy array (NOT a numpy array subclasses).
- xarray **HAS A** numpy array (it is not an "IS A" relationship)
- Often have to extract the numpy array from the xarray array before passing it to extension modules
- Most numpy methods are available in xarray, but not all.

11.1 Creating an xarray Array from a numpy Array

```
import numpy
import xarray

my_array = numpy.zeros((3,3,3), "float32")
```

```
# Making up dimension names and
# coordinates.
my_name = "my_xarray"

my_dims = ["bottom_top", "south_north",
           "west_east"]

my_coords = {"bottom_top" :
                  [100., 200., 300.],
          "south_north":
                  [40., 50., 60.],
          "west_east" :
                  [-120., -110., -100.]
         }

my_attrs = {"info" : "This is my xarray"}

my_xarray = xarray.DataArray(my_array,
                        name=my_name,
                        dims=my_dims,
                        coords=my_coords,
                        attrs=my_attrs)
```

11.2 Your Turn!

11.2.1 Example 1.3: Creating an xarray DataArray

```
In [3]: import numpy
        import xarray

        my_array = numpy.zeros((3,3,3), "float32")

        # Making up dimension names and
        # coordinates.
        my_name = "my_xarray"

        my_dims = ["bottom_top", "south_north", "west_east"]

        my_coords = {"bottom_top" : [100., 200., 300.],
                  "south_north": [40., 50., 60.],
                  "west_east" : [-120., -110., -100.]
                 }

        my_attrs = {"info" : "This is my xarray"}

        my_xarray = xarray.DataArray(my_array,
                                     name=my_name,
                                     dims=my_dims,
```

```
                                    coords=my_coords,
                                    attrs=my_attrs)

        print (my_xarray)
```

```
<xarray.DataArray 'my_xarray' (bottom_top: 3, south_north: 3, west_east: 3)>
array([[[ 0.,  0.,  0.],
        [ 0.,  0.,  0.],
        [ 0.,  0.,  0.]],

       [[ 0.,  0.,  0.],
        [ 0.,  0.,  0.],
        [ 0.,  0.,  0.]],

       [[ 0.,  0.,  0.],
        [ 0.,  0.,  0.],
        [ 0.,  0.,  0.]]], dtype=float32)
Coordinates:
  * south_north  (south_north) float64 40.0 50.0 60.0
  * west_east    (west_east) float64 -120.0 -110.0 -100.0
  * bottom_top   (bottom_top) float64 100.0 200.0 300.0
Attributes:
    info:     This is my xarray
```

11.3 xarray and Missing Data Values

- xarray always uses NaN for missing data values.
- Can cause problems with compiled numerical routines.
- Can cause problems for algorithms expecting MaskedArrays.
- wrf-python includes the fill value information in the attr section of the metadata (_FillValue).
- The *to_np* routine can be used to convert xarray arrays to numpy/masked arrays.

11.4 Your Turn!

11.5 Example 1.4: xarray and Missing Values

```
In [4]: import numpy
        import numpy.ma
        import xarray

        from wrf import to_np

        # Create a MaskedArray with 10.0 in the center
        my_array = numpy.zeros((3,3,3), "float32")

        my_array[1,1,1] = 10.0

        my_masked = numpy.ma.masked_equal(my_array, 0)
```

```
        # Making up dimension names and
        # coordinates.
        my_name = "my_masked_xarray"

        my_dims = ["bottom_top", "south_north", "west_east"]

        my_coords = {"bottom_top" : [100., 200., 300.],
                     "south_north": [40., 50., 60.],
                     "west_east" : [-120., -110., -100.]
                    }

        my_attrs = {"info" : "This is my masked xarray",
                    "_FillValue" : -999.0}

        # Create the xarray DataArray
        my_xarray = xarray.DataArray(my_masked,
                                     name=my_name,
                                     dims=my_dims,
                                     coords=my_coords,
                                     attrs=my_attrs)

        print ("xarray Array with Missing Values")
        print (my_xarray)
        print ("\n")

        # Covert back to a MaskedArray
        converted = to_np(my_xarray)

        print ("Converted to a MaskedArray with to_np")
        print (converted)
```

```
xarray Array with Missing Values
<xarray.DataArray 'my_masked_xarray' (bottom_top: 3, south_north: 3, west_east: 3)>
array([[[ nan,  nan,  nan],
        [ nan,  nan,  nan],
        [ nan,  nan,  nan]],

       [[ nan,  nan,  nan],
        [ nan,  10.,  nan],
        [ nan,  nan,  nan]],

       [[ nan,  nan,  nan],
        [ nan,  nan,  nan],
        [ nan,  nan,  nan]]], dtype=float32)
Coordinates:
  * south_north  (south_north) float64 40.0 50.0 60.0
  * west_east    (west_east) float64 -120.0 -110.0 -100.0
```

```
  * bottom_top   (bottom_top) float64 100.0 200.0 300.0
Attributes:
    info:        This is my masked xarray
    _FillValue:  -999.0


Converted to a MaskedArray with to_np
[[[-- -- --]
  [-- -- --]
  [-- -- --]]

 [[-- -- --]
  [-- 10.0 --]
  [-- -- --]]

 [[-- -- --]
  [-- -- --]
  [-- -- --]]]
```

12 2.0 Overview of WRF Output Data

The first rule of data processing:
"ALWAYS LOOK AT YOUR DATA"
- D. Shea

12.1 Why Look At WRF Data? Isn't It All the Same?

- WRF can be configured in various ways and can have variables turned on and off.
- If you run in to problems, it could be due to a variable missing.
- If your plot doesn't look right, there could be a map projection issue.

12.2 Data Viewing Tools

There are numerous tools available to examine NetCDF data, from both outside and inside of Python.

- **ncdump** (used for this example)
- ncl_filedump
- netcdf4-python
- PyNIO
- xarray

12.3 ncdump

ncdump is a program included with the NetCDF libraries that can be used to examine NetCDF data.

By supplying the '-h' option, only the data descriptions are returned. Otherwise, you'll get all of the data values, which can span miles.

To run:

```
$ ncdump -h wrfout_d01_2005-08-28_00:00:00
```

12.4 ncdump Output

```
netcdf wrfout_d01_2005-08-28_00\:00\:00 {
dimensions:
    Time = UNLIMITED ; // (4 currently)

    DateStrLen = 19 ;

    west_east = 90 ;

    south_north = 73 ;

    bottom_top = 29 ;

    bottom_top_stag = 30 ;

    soil_layers_stag = 4 ;

    west_east_stag = 91 ;

    south_north_stag = 74 ;
```

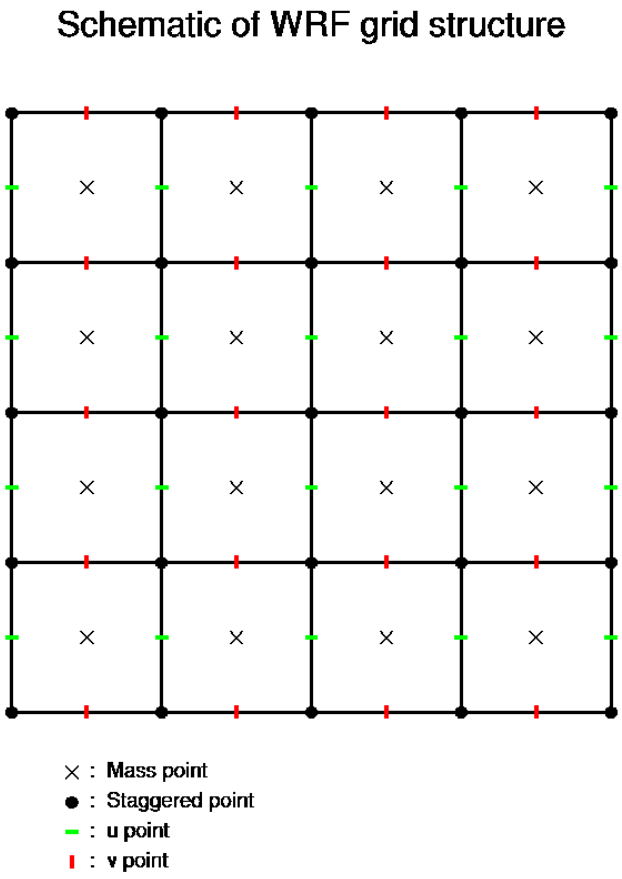
12.5 ncdump Output

```
variables:
    char Times(Time, DateStrLen) ;
    float XLAT(Time, south_north, west_east) ;
        XLAT:FieldType = 104 ;
        XLAT:MemoryOrder = "XY " ;
        XLAT:description = "LATITUDE, SOUTH IS NEGATIVE" ;
        XLAT:units = "degree_north" ;
        XLAT:stagger = "" ;
        XLAT:coordinates = "XLONG XLAT" ;
    float XLONG(Time, south_north, west_east) ;
        XLONG:FieldType = 104 ;
        XLONG:MemoryOrder = "XY " ;
        XLONG:description = "LONGITUDE, WEST IS NEGATIVE" ;
        XLONG:units = "degree_east" ;
```

```
        XLONG:stagger = "" ;
        XLONG:coordinates = "XLONG XLAT" ;
        .
        .
        .
    float SST_INPUT(Time, south_north, west_east) ;
        SST_INPUT:FieldType = 104 ;
        SST_INPUT:MemoryOrder = "XY " ;
        SST_INPUT:description = "SEA SURFACE TEMPERATURE
            FROM WRFLOWINPUT FILE" ;
        SST_INPUT:units = "K" ;
        SST_INPUT:stagger = "" ;
        SST_INPUT:coordinates = "XLONG XLAT XTIME" ;
```

12.6 ncdump Output

```
// global attributes:
        :TITLE = " OUTPUT FROM WRF V3.7 MODEL" ;
        :START_DATE = "2005-08-28_00:00:00" ;
        :SIMULATION_START_DATE = "2005-08-28_00:00:00" ;
        :WEST-EAST_GRID_DIMENSION = 91 ;
        :SOUTH-NORTH_GRID_DIMENSION = 74 ;
        :BOTTOM-TOP_GRID_DIMENSION = 30 ;
        :DX = 30000.f ;
        :DY = 30000.f ;
        .
        .
        .
        :CEN_LAT = 28.00002f ;
        :CEN_LON = -89.f ;
        :TRUELAT1 = 30.f ;
        :TRUELAT2 = 60.f ;
        :MOAD_CEN_LAT = 28.00002f ;
        :STAND_LON = -89.f ;
        :POLE_LAT = 90.f ;
        :POLE_LON = 0.f ;
        :GMT = 0.f ;
        :JULYR = 2005 ;
        :JULDAY = 240 ;
        :MAP_PROJ = 1 ;
        :MAP_PROJ_CHAR = "Lambert Conformal" ;
        .
        .
        .
}
```



alt

12.7 Dimensions

WRF-ARW uses an Arakawa C-grid staggered grid (taken from mmm website)

- Mass related quantities (pressure, temperature, etc) are computed at the center of a grid cell.
- The u-component of the horizontal wind is calculated at the left and right edges of a grid cell. It has one more point in the x direction than the mass grid.
- The v-component of the horizontal wind is calculated at the bottom and top edges of a grid cell. It has one more point in the y direction than the mass grid.
- The corners of each grid box are know as the 'staggered' grid, and has one additional point in both the x and y direction.

```
netcdf wrfout_d01_2005-08-28_00\:00\:00 {
dimensions:
    Time = UNLIMITED ; // (4 currently)

    DateStrLen = 19 ;
```

```
    west_east = 90 ;

    south_north = 73 ;

    bottom_top = 29 ;

    bottom_top_stag = 30 ; <-- Extra grid point

    soil_layers_stag = 4 ;

    west_east_stag = 91 ; <-- Extra grid point

    south_north_stag = 74 ; <-- Extra grid point
```

12.8 Variables

- Each variable is made up of dimensions, attributes, and data values.
- Pay special attention to the units and coordinates attribute.
- The *coordinates* attribute specifies the variables that contain the latitude and longitude information for each grid box (XLONG, XLAT).
- More recent versions of WRF include an XTIME coordinate.
- The coordinates are named with Fortran ordering, so they'll be listed in reverse.

```
float P(Time, bottom_top, south_north, west_east) ; <- Dims
    P:FieldType = 104 ;                       <-  Attribute
    P:MemoryOrder = "XYZ" ;                   <-  Attribute
    P:description = "perturbation pressure" ; <-  Attribute
    P:units = "Pa" ;                          <-  Attribute
    P:stagger = "" ;                          <-  Attribute
    P:coordinates = "XLONG XLAT XTIME" ;      <-  Attribute

data:

 P =
  339.8281, 340.3281, 340.25, 341.4531, ...
    355.8672, 356.9531, 361.2578, 365.7188,  ...
```

12.9 Global Attributes

- Provide a description of how the model was set up (resolution, map projection, microphysics, etc)
- For plotting, the map projection parameters will be the most important.
- wrf-python uses this information to build the mapping object in your plotting system of choice - basemap, cartopy, pyngl.

.

```
.
.
:CEN_LAT = 28.00002f ;
:CEN_LON = -89.f ;
:TRUELAT1 = 30.f ;
:TRUELAT2 = 60.f ;
:MOAD_CEN_LAT = 28.00002f ;
:STAND_LON = -89.f ;
:POLE_LAT = 90.f ;
:POLE_LON = 0.f ;
.
.
:MAP_PROJ = 1 ;
:MAP_PROJ_CHAR = "Lambert Conformal" ;
.
.
.
```

12.10 Your Turn!

12.10.1 Example 2.1: Running ncdump

```
In [5]: from subprocess import Popen, PIPE, STDOUT

        file_path = single_wrf_file()

        # This simply executes 'ncdump -h {wrf_file}'
        # from Python
        p = Popen(["ncdump", "-h", "{}".format(file_path)],
                  stdout=PIPE, stderr=STDOUT)
        output, _ = p.communicate()

        print (output)
```

```
netcdf wrfout_d01_2005-08-28_00_00_00 {
dimensions:
        Time = UNLIMITED ; // (4 currently)
        DateStrLen = 19 ;
        west_east = 90 ;
        south_north = 73 ;
        bottom_top = 29 ;
        bottom_top_stag = 30 ;
        soil_layers_stag = 4 ;
        west_east_stag = 91 ;
        south_north_stag = 74 ;
variables:
        char Times(Time, DateStrLen) ;
        float XLAT(Time, south_north, west_east) ;
                XLAT:FieldType = 104 ;
```

```
        XLAT:MemoryOrder = "XY " ;
        XLAT:description = "LATITUDE, SOUTH IS NEGATIVE" ;
        XLAT:units = "degree_north" ;
        XLAT:stagger = "" ;
        XLAT:coordinates = "XLONG XLAT" ;
float XLONG(Time, south_north, west_east) ;
        XLONG:FieldType = 104 ;
        XLONG:MemoryOrder = "XY " ;
        XLONG:description = "LONGITUDE, WEST IS NEGATIVE" ;
        XLONG:units = "degree_east" ;
        XLONG:stagger = "" ;
        XLONG:coordinates = "XLONG XLAT" ;
float LU_INDEX(Time, south_north, west_east) ;
        LU_INDEX:FieldType = 104 ;
        LU_INDEX:MemoryOrder = "XY " ;
        LU_INDEX:description = "LAND USE CATEGORY" ;
        LU_INDEX:units = "" ;
        LU_INDEX:stagger = "" ;
        LU_INDEX:coordinates = "XLONG XLAT XTIME" ;
float ZNU(Time, bottom_top) ;
        ZNU:FieldType = 104 ;
        ZNU:MemoryOrder = "Z  " ;
        ZNU:description = "eta values on half (mass) levels" ;
        ZNU:units = "" ;
        ZNU:stagger = "" ;
float ZNW(Time, bottom_top_stag) ;
        ZNW:FieldType = 104 ;
        ZNW:MemoryOrder = "Z  " ;
        ZNW:description = "eta values on full (w) levels" ;
        ZNW:units = "" ;
        ZNW:stagger = "Z" ;
float ZS(Time, soil_layers_stag) ;
        ZS:FieldType = 104 ;
        ZS:MemoryOrder = "Z  " ;
        ZS:description = "DEPTHS OF CENTERS OF SOIL LAYERS" ;
        ZS:units = "m" ;
        ZS:stagger = "Z" ;
float DZS(Time, soil_layers_stag) ;
        DZS:FieldType = 104 ;
        DZS:MemoryOrder = "Z  " ;
        DZS:description = "THICKNESSES OF SOIL LAYERS" ;
        DZS:units = "m" ;
        DZS:stagger = "Z" ;
float VAR_SSO(Time, south_north, west_east) ;
        VAR_SSO:FieldType = 104 ;
        VAR_SSO:MemoryOrder = "XY " ;
        VAR_SSO:description = "variance of subgrid-scale orography" ;
        VAR_SSO:units = "m2" ;
```

```
        VAR_SSO:stagger = "" ;
        VAR_SSO:coordinates = "XLONG XLAT XTIME" ;
float LAP_HGT(Time, south_north, west_east) ;
        LAP_HGT:FieldType = 104 ;
        LAP_HGT:MemoryOrder = "XY " ;
        LAP_HGT:description = "Laplacian of orography" ;
        LAP_HGT:units = "m" ;
        LAP_HGT:stagger = "" ;
        LAP_HGT:coordinates = "XLONG XLAT XTIME" ;
float U(Time, bottom_top, south_north, west_east_stag) ;
        U:FieldType = 104 ;
        U:MemoryOrder = "XYZ" ;
        U:description = "x-wind component" ;
        U:units = "m s-1" ;
        U:stagger = "X" ;
        U:coordinates = "XLONG_U XLAT_U XTIME" ;
float V(Time, bottom_top, south_north_stag, west_east) ;
        V:FieldType = 104 ;
        V:MemoryOrder = "XYZ" ;
        V:description = "y-wind component" ;
        V:units = "m s-1" ;
        V:stagger = "Y" ;
        V:coordinates = "XLONG_V XLAT_V XTIME" ;
float W(Time, bottom_top_stag, south_north, west_east) ;
        W:FieldType = 104 ;
        W:MemoryOrder = "XYZ" ;
        W:description = "z-wind component" ;
        W:units = "m s-1" ;
        W:stagger = "Z" ;
        W:coordinates = "XLONG XLAT XTIME" ;
float PH(Time, bottom_top_stag, south_north, west_east) ;
        PH:FieldType = 104 ;
        PH:MemoryOrder = "XYZ" ;
        PH:description = "perturbation geopotential" ;
        PH:units = "m2 s-2" ;
        PH:stagger = "Z" ;
        PH:coordinates = "XLONG XLAT XTIME" ;
float PHB(Time, bottom_top_stag, south_north, west_east) ;
        PHB:FieldType = 104 ;
        PHB:MemoryOrder = "XYZ" ;
        PHB:description = "base-state geopotential" ;
        PHB:units = "m2 s-2" ;
        PHB:stagger = "Z" ;
        PHB:coordinates = "XLONG XLAT XTIME" ;
float T(Time, bottom_top, south_north, west_east) ;
        T:FieldType = 104 ;
        T:MemoryOrder = "XYZ" ;
        T:description = "perturbation potential temperature (theta-t0)" ;
```

```
        T:units = "K" ;
        T:stagger = "" ;
        T:coordinates = "XLONG XLAT XTIME" ;
float HFX_FORCE(Time) ;
        HFX_FORCE:FieldType = 104 ;
        HFX_FORCE:MemoryOrder = "0  " ;
        HFX_FORCE:description = "SCM ideal surface sensible heat flux" ;
        HFX_FORCE:units = "W m-2" ;
        HFX_FORCE:stagger = "" ;
float LH_FORCE(Time) ;
        LH_FORCE:FieldType = 104 ;
        LH_FORCE:MemoryOrder = "0  " ;
        LH_FORCE:description = "SCM ideal surface latent heat flux" ;
        LH_FORCE:units = "W m-2" ;
        LH_FORCE:stagger = "" ;
float TSK_FORCE(Time) ;
        TSK_FORCE:FieldType = 104 ;
        TSK_FORCE:MemoryOrder = "0  " ;
        TSK_FORCE:description = "SCM ideal surface skin temperature" ;
        TSK_FORCE:units = "W m-2" ;
        TSK_FORCE:stagger = "" ;
float HFX_FORCE_TEND(Time) ;
        HFX_FORCE_TEND:FieldType = 104 ;
        HFX_FORCE_TEND:MemoryOrder = "0  " ;
        HFX_FORCE_TEND:description = "SCM ideal surface sensible heat flux tendency" ;
        HFX_FORCE_TEND:units = "W m-2 s-1" ;
        HFX_FORCE_TEND:stagger = "" ;
float LH_FORCE_TEND(Time) ;
        LH_FORCE_TEND:FieldType = 104 ;
        LH_FORCE_TEND:MemoryOrder = "0  " ;
        LH_FORCE_TEND:description = "SCM ideal surface latent heat flux tendency" ;
        LH_FORCE_TEND:units = "W m-2 s-1" ;
        LH_FORCE_TEND:stagger = "" ;
float TSK_FORCE_TEND(Time) ;
        TSK_FORCE_TEND:FieldType = 104 ;
        TSK_FORCE_TEND:MemoryOrder = "0  " ;
        TSK_FORCE_TEND:description = "SCM ideal surface skin temperature tendency" ;
        TSK_FORCE_TEND:units = "W m-2 s-1" ;
        TSK_FORCE_TEND:stagger = "" ;
float MU(Time, south_north, west_east) ;
        MU:FieldType = 104 ;
        MU:MemoryOrder = "XY " ;
        MU:description = "perturbation dry air mass in column" ;
        MU:units = "Pa" ;
        MU:stagger = "" ;
        MU:coordinates = "XLONG XLAT XTIME" ;
float MUB(Time, south_north, west_east) ;
        MUB:FieldType = 104 ;
```

```
        MUB:MemoryOrder = "XY " ;
        MUB:description = "base state dry air mass in column" ;
        MUB:units = "Pa" ;
        MUB:stagger = "" ;
        MUB:coordinates = "XLONG XLAT XTIME" ;
float NEST_POS(Time, south_north, west_east) ;
        NEST_POS:FieldType = 104 ;
        NEST_POS:MemoryOrder = "XY " ;
        NEST_POS:description = "-" ;
        NEST_POS:units = "-" ;
        NEST_POS:stagger = "" ;
        NEST_POS:coordinates = "XLONG XLAT XTIME" ;
float P(Time, bottom_top, south_north, west_east) ;
        P:FieldType = 104 ;
        P:MemoryOrder = "XYZ" ;
        P:description = "perturbation pressure" ;
        P:units = "Pa" ;
        P:stagger = "" ;
        P:coordinates = "XLONG XLAT XTIME" ;
float PB(Time, bottom_top, south_north, west_east) ;
        PB:FieldType = 104 ;
        PB:MemoryOrder = "XYZ" ;
        PB:description = "BASE STATE PRESSURE" ;
        PB:units = "Pa" ;
        PB:stagger = "" ;
        PB:coordinates = "XLONG XLAT XTIME" ;
float FNM(Time, bottom_top) ;
        FNM:FieldType = 104 ;
        FNM:MemoryOrder = "Z  " ;
        FNM:description = "upper weight for vertical stretching" ;
        FNM:units = "" ;
        FNM:stagger = "" ;
float FNP(Time, bottom_top) ;
        FNP:FieldType = 104 ;
        FNP:MemoryOrder = "Z  " ;
        FNP:description = "lower weight for vertical stretching" ;
        FNP:units = "" ;
        FNP:stagger = "" ;
float RDNW(Time, bottom_top) ;
        RDNW:FieldType = 104 ;
        RDNW:MemoryOrder = "Z  " ;
        RDNW:description = "inverse d(eta) values between full (w) levels" ;
        RDNW:units = "" ;
        RDNW:stagger = "" ;
float RDN(Time, bottom_top) ;
        RDN:FieldType = 104 ;
        RDN:MemoryOrder = "Z  " ;
        RDN:description = "inverse d(eta) values between half (mass) levels" ;
```

```
        RDN:units = "" ;
        RDN:stagger = "" ;
float DNW(Time, bottom_top) ;
        DNW:FieldType = 104 ;
        DNW:MemoryOrder = "Z  " ;
        DNW:description = "d(eta) values between full (w) levels" ;
        DNW:units = "" ;
        DNW:stagger = "" ;
float DN(Time, bottom_top) ;
        DN:FieldType = 104 ;
        DN:MemoryOrder = "Z  " ;
        DN:description = "d(eta) values between half (mass) levels" ;
        DN:units = "" ;
        DN:stagger = "" ;
float CFN(Time) ;
        CFN:FieldType = 104 ;
        CFN:MemoryOrder = "0  " ;
        CFN:description = "extrapolation constant" ;
        CFN:units = "" ;
        CFN:stagger = "" ;
float CFN1(Time) ;
        CFN1:FieldType = 104 ;
        CFN1:MemoryOrder = "0  " ;
        CFN1:description = "extrapolation constant" ;
        CFN1:units = "" ;
        CFN1:stagger = "" ;
int THIS_IS_AN_IDEAL_RUN(Time) ;
        THIS_IS_AN_IDEAL_RUN:FieldType = 106 ;
        THIS_IS_AN_IDEAL_RUN:MemoryOrder = "0  " ;
        THIS_IS_AN_IDEAL_RUN:description = "T/F flag: this is an ARW ideal simulation"
        THIS_IS_AN_IDEAL_RUN:units = "-" ;
        THIS_IS_AN_IDEAL_RUN:stagger = "" ;
float P_HYD(Time, bottom_top, south_north, west_east) ;
        P_HYD:FieldType = 104 ;
        P_HYD:MemoryOrder = "XYZ" ;
        P_HYD:description = "hydrostatic pressure" ;
        P_HYD:units = "Pa" ;
        P_HYD:stagger = "" ;
        P_HYD:coordinates = "XLONG XLAT XTIME" ;
float Q2(Time, south_north, west_east) ;
        Q2:FieldType = 104 ;
        Q2:MemoryOrder = "XY " ;
        Q2:description = "QV at 2 M" ;
        Q2:units = "kg kg-1" ;
        Q2:stagger = "" ;
        Q2:coordinates = "XLONG XLAT XTIME" ;
float T2(Time, south_north, west_east) ;
        T2:FieldType = 104 ;
```

```
        T2:MemoryOrder = "XY " ;
        T2:description = "TEMP at 2 M" ;
        T2:units = "K" ;
        T2:stagger = "" ;
        T2:coordinates = "XLONG XLAT XTIME" ;
float TH2(Time, south_north, west_east) ;
        TH2:FieldType = 104 ;
        TH2:MemoryOrder = "XY " ;
        TH2:description = "POT TEMP at 2 M" ;
        TH2:units = "K" ;
        TH2:stagger = "" ;
        TH2:coordinates = "XLONG XLAT XTIME" ;
float PSFC(Time, south_north, west_east) ;
        PSFC:FieldType = 104 ;
        PSFC:MemoryOrder = "XY " ;
        PSFC:description = "SFC PRESSURE" ;
        PSFC:units = "Pa" ;
        PSFC:stagger = "" ;
        PSFC:coordinates = "XLONG XLAT XTIME" ;
float U10(Time, south_north, west_east) ;
        U10:FieldType = 104 ;
        U10:MemoryOrder = "XY " ;
        U10:description = "U at 10 M" ;
        U10:units = "m s-1" ;
        U10:stagger = "" ;
        U10:coordinates = "XLONG XLAT XTIME" ;
float V10(Time, south_north, west_east) ;
        V10:FieldType = 104 ;
        V10:MemoryOrder = "XY " ;
        V10:description = "V at 10 M" ;
        V10:units = "m s-1" ;
        V10:stagger = "" ;
        V10:coordinates = "XLONG XLAT XTIME" ;
float RDX(Time) ;
        RDX:FieldType = 104 ;
        RDX:MemoryOrder = "0  " ;
        RDX:description = "INVERSE X GRID LENGTH" ;
        RDX:units = "" ;
        RDX:stagger = "" ;
float RDY(Time) ;
        RDY:FieldType = 104 ;
        RDY:MemoryOrder = "0  " ;
        RDY:description = "INVERSE Y GRID LENGTH" ;
        RDY:units = "" ;
        RDY:stagger = "" ;
float RESM(Time) ;
        RESM:FieldType = 104 ;
        RESM:MemoryOrder = "0  " ;
```

```
        RESM:description = "TIME WEIGHT CONSTANT FOR SMALL STEPS" ;
        RESM:units = "" ;
        RESM:stagger = "" ;
float ZETATOP(Time) ;
        ZETATOP:FieldType = 104 ;
        ZETATOP:MemoryOrder = "0  " ;
        ZETATOP:description = "ZETA AT MODEL TOP" ;
        ZETATOP:units = "" ;
        ZETATOP:stagger = "" ;
float CF1(Time) ;
        CF1:FieldType = 104 ;
        CF1:MemoryOrder = "0  " ;
        CF1:description = "2nd order extrapolation constant" ;
        CF1:units = "" ;
        CF1:stagger = "" ;
float CF2(Time) ;
        CF2:FieldType = 104 ;
        CF2:MemoryOrder = "0  " ;
        CF2:description = "2nd order extrapolation constant" ;
        CF2:units = "" ;
        CF2:stagger = "" ;
float CF3(Time) ;
        CF3:FieldType = 104 ;
        CF3:MemoryOrder = "0  " ;
        CF3:description = "2nd order extrapolation constant" ;
        CF3:units = "" ;
        CF3:stagger = "" ;
int ITIMESTEP(Time) ;
        ITIMESTEP:FieldType = 106 ;
        ITIMESTEP:MemoryOrder = "0  " ;
        ITIMESTEP:description = "" ;
        ITIMESTEP:units = "" ;
        ITIMESTEP:stagger = "" ;
float XTIME(Time) ;
        XTIME:FieldType = 104 ;
        XTIME:MemoryOrder = "0  " ;
        XTIME:description = "minutes since 2005-08-28 00:00:00" ;
        XTIME:units = "minutes since 2005-08-28 00:00:00" ;
        XTIME:stagger = "" ;
float QVAPOR(Time, bottom_top, south_north, west_east) ;
        QVAPOR:FieldType = 104 ;
        QVAPOR:MemoryOrder = "XYZ" ;
        QVAPOR:description = "Water vapor mixing ratio" ;
        QVAPOR:units = "kg kg-1" ;
        QVAPOR:stagger = "" ;
        QVAPOR:coordinates = "XLONG XLAT XTIME" ;
float QCLOUD(Time, bottom_top, south_north, west_east) ;
        QCLOUD:FieldType = 104 ;
```

```
        QCLOUD:MemoryOrder = "XYZ" ;
        QCLOUD:description = "Cloud water mixing ratio" ;
        QCLOUD:units = "kg kg-1" ;
        QCLOUD:stagger = "" ;
        QCLOUD:coordinates = "XLONG XLAT XTIME" ;
float QRAIN(Time, bottom_top, south_north, west_east) ;
        QRAIN:FieldType = 104 ;
        QRAIN:MemoryOrder = "XYZ" ;
        QRAIN:description = "Rain water mixing ratio" ;
        QRAIN:units = "kg kg-1" ;
        QRAIN:stagger = "" ;
        QRAIN:coordinates = "XLONG XLAT XTIME" ;
float SHDMAX(Time, south_north, west_east) ;
        SHDMAX:FieldType = 104 ;
        SHDMAX:MemoryOrder = "XY " ;
        SHDMAX:description = "ANNUAL MAX VEG FRACTION" ;
        SHDMAX:units = "" ;
        SHDMAX:stagger = "" ;
        SHDMAX:coordinates = "XLONG XLAT XTIME" ;
float SHDMIN(Time, south_north, west_east) ;
        SHDMIN:FieldType = 104 ;
        SHDMIN:MemoryOrder = "XY " ;
        SHDMIN:description = "ANNUAL MIN VEG FRACTION" ;
        SHDMIN:units = "" ;
        SHDMIN:stagger = "" ;
        SHDMIN:coordinates = "XLONG XLAT XTIME" ;
float SNOALB(Time, south_north, west_east) ;
        SNOALB:FieldType = 104 ;
        SNOALB:MemoryOrder = "XY " ;
        SNOALB:description = "ANNUAL MAX SNOW ALBEDO IN FRACTION" ;
        SNOALB:units = "" ;
        SNOALB:stagger = "" ;
        SNOALB:coordinates = "XLONG XLAT XTIME" ;
float TSLB(Time, soil_layers_stag, south_north, west_east) ;
        TSLB:FieldType = 104 ;
        TSLB:MemoryOrder = "XYZ" ;
        TSLB:description = "SOIL TEMPERATURE" ;
        TSLB:units = "K" ;
        TSLB:stagger = "Z" ;
        TSLB:coordinates = "XLONG XLAT XTIME" ;
float SMOIS(Time, soil_layers_stag, south_north, west_east) ;
        SMOIS:FieldType = 104 ;
        SMOIS:MemoryOrder = "XYZ" ;
        SMOIS:description = "SOIL MOISTURE" ;
        SMOIS:units = "m3 m-3" ;
        SMOIS:stagger = "Z" ;
        SMOIS:coordinates = "XLONG XLAT XTIME" ;
float SH2O(Time, soil_layers_stag, south_north, west_east) ;
```

```
        SH2O:FieldType = 104 ;
        SH2O:MemoryOrder = "XYZ" ;
        SH2O:description = "SOIL LIQUID WATER" ;
        SH2O:units = "m3 m-3" ;
        SH2O:stagger = "Z" ;
        SH2O:coordinates = "XLONG XLAT XTIME" ;
float SMCREL(Time, soil_layers_stag, south_north, west_east) ;
        SMCREL:FieldType = 104 ;
        SMCREL:MemoryOrder = "XYZ" ;
        SMCREL:description = "RELATIVE SOIL MOISTURE" ;
        SMCREL:units = "" ;
        SMCREL:stagger = "Z" ;
        SMCREL:coordinates = "XLONG XLAT XTIME" ;
float SEAICE(Time, south_north, west_east) ;
        SEAICE:FieldType = 104 ;
        SEAICE:MemoryOrder = "XY " ;
        SEAICE:description = "SEA ICE FLAG" ;
        SEAICE:units = "" ;
        SEAICE:stagger = "" ;
        SEAICE:coordinates = "XLONG XLAT XTIME" ;
float XICEM(Time, south_north, west_east) ;
        XICEM:FieldType = 104 ;
        XICEM:MemoryOrder = "XY " ;
        XICEM:description = "SEA ICE FLAG (PREVIOUS STEP)" ;
        XICEM:units = "" ;
        XICEM:stagger = "" ;
        XICEM:coordinates = "XLONG XLAT XTIME" ;
float SFROFF(Time, south_north, west_east) ;
        SFROFF:FieldType = 104 ;
        SFROFF:MemoryOrder = "XY " ;
        SFROFF:description = "SURFACE RUNOFF" ;
        SFROFF:units = "mm" ;
        SFROFF:stagger = "" ;
        SFROFF:coordinates = "XLONG XLAT XTIME" ;
float UDROFF(Time, south_north, west_east) ;
        UDROFF:FieldType = 104 ;
        UDROFF:MemoryOrder = "XY " ;
        UDROFF:description = "UNDERGROUND RUNOFF" ;
        UDROFF:units = "mm" ;
        UDROFF:stagger = "" ;
        UDROFF:coordinates = "XLONG XLAT XTIME" ;
int IVGTYP(Time, south_north, west_east) ;
        IVGTYP:FieldType = 106 ;
        IVGTYP:MemoryOrder = "XY " ;
        IVGTYP:description = "DOMINANT VEGETATION CATEGORY" ;
        IVGTYP:units = "" ;
        IVGTYP:stagger = "" ;
        IVGTYP:coordinates = "XLONG XLAT XTIME" ;
```

```
int ISLTYP(Time, south_north, west_east) ;
        ISLTYP:FieldType = 106 ;
        ISLTYP:MemoryOrder = "XY " ;
        ISLTYP:description = "DOMINANT SOIL CATEGORY" ;
        ISLTYP:units = "" ;
        ISLTYP:stagger = "" ;
        ISLTYP:coordinates = "XLONG XLAT XTIME" ;
float VEGFRA(Time, south_north, west_east) ;
        VEGFRA:FieldType = 104 ;
        VEGFRA:MemoryOrder = "XY " ;
        VEGFRA:description = "VEGETATION FRACTION" ;
        VEGFRA:units = "" ;
        VEGFRA:stagger = "" ;
        VEGFRA:coordinates = "XLONG XLAT XTIME" ;
float GRDFLX(Time, south_north, west_east) ;
        GRDFLX:FieldType = 104 ;
        GRDFLX:MemoryOrder = "XY " ;
        GRDFLX:description = "GROUND HEAT FLUX" ;
        GRDFLX:units = "W m-2" ;
        GRDFLX:stagger = "" ;
        GRDFLX:coordinates = "XLONG XLAT XTIME" ;
float ACGRDFLX(Time, south_north, west_east) ;
        ACGRDFLX:FieldType = 104 ;
        ACGRDFLX:MemoryOrder = "XY " ;
        ACGRDFLX:description = "ACCUMULATED GROUND HEAT FLUX" ;
        ACGRDFLX:units = "J m-2" ;
        ACGRDFLX:stagger = "" ;
        ACGRDFLX:coordinates = "XLONG XLAT XTIME" ;
float ACSNOM(Time, south_north, west_east) ;
        ACSNOM:FieldType = 104 ;
        ACSNOM:MemoryOrder = "XY " ;
        ACSNOM:description = "ACCUMULATED MELTED SNOW" ;
        ACSNOM:units = "kg m-2" ;
        ACSNOM:stagger = "" ;
        ACSNOM:coordinates = "XLONG XLAT XTIME" ;
float SNOW(Time, south_north, west_east) ;
        SNOW:FieldType = 104 ;
        SNOW:MemoryOrder = "XY " ;
        SNOW:description = "SNOW WATER EQUIVALENT" ;
        SNOW:units = "kg m-2" ;
        SNOW:stagger = "" ;
        SNOW:coordinates = "XLONG XLAT XTIME" ;
float SNOWH(Time, south_north, west_east) ;
        SNOWH:FieldType = 104 ;
        SNOWH:MemoryOrder = "XY " ;
        SNOWH:description = "PHYSICAL SNOW DEPTH" ;
        SNOWH:units = "m" ;
        SNOWH:stagger = "" ;
```

```
        SNOWH:coordinates = "XLONG XLAT XTIME" ;
float CANWAT(Time, south_north, west_east) ;
        CANWAT:FieldType = 104 ;
        CANWAT:MemoryOrder = "XY " ;
        CANWAT:description = "CANOPY WATER" ;
        CANWAT:units = "kg m-2" ;
        CANWAT:stagger = "" ;
        CANWAT:coordinates = "XLONG XLAT XTIME" ;
float SSTSK(Time, south_north, west_east) ;
        SSTSK:FieldType = 104 ;
        SSTSK:MemoryOrder = "XY " ;
        SSTSK:description = "SKIN SEA SURFACE TEMPERATURE" ;
        SSTSK:units = "K" ;
        SSTSK:stagger = "" ;
        SSTSK:coordinates = "XLONG XLAT XTIME" ;
float COSZEN(Time, south_north, west_east) ;
        COSZEN:FieldType = 104 ;
        COSZEN:MemoryOrder = "XY " ;
        COSZEN:description = "COS of SOLAR ZENITH ANGLE" ;
        COSZEN:units = "dimensionless" ;
        COSZEN:stagger = "" ;
        COSZEN:coordinates = "XLONG XLAT XTIME" ;
float LAI(Time, south_north, west_east) ;
        LAI:FieldType = 104 ;
        LAI:MemoryOrder = "XY " ;
        LAI:description = "LEAF AREA INDEX" ;
        LAI:units = "m-2/m-2" ;
        LAI:stagger = "" ;
        LAI:coordinates = "XLONG XLAT XTIME" ;
float VAR(Time, south_north, west_east) ;
        VAR:FieldType = 104 ;
        VAR:MemoryOrder = "XY " ;
        VAR:description = "OROGRAPHIC VARIANCE" ;
        VAR:units = "" ;
        VAR:stagger = "" ;
        VAR:coordinates = "XLONG XLAT XTIME" ;
float MAPFAC_M(Time, south_north, west_east) ;
        MAPFAC_M:FieldType = 104 ;
        MAPFAC_M:MemoryOrder = "XY " ;
        MAPFAC_M:description = "Map scale factor on mass grid" ;
        MAPFAC_M:units = "" ;
        MAPFAC_M:stagger = "" ;
        MAPFAC_M:coordinates = "XLONG XLAT XTIME" ;
float MAPFAC_U(Time, south_north, west_east_stag) ;
        MAPFAC_U:FieldType = 104 ;
        MAPFAC_U:MemoryOrder = "XY " ;
        MAPFAC_U:description = "Map scale factor on u-grid" ;
        MAPFAC_U:units = "" ;
```

```
        MAPFAC_U:stagger = "X" ;
        MAPFAC_U:coordinates = "XLONG_U XLAT_U XTIME" ;
float MAPFAC_V(Time, south_north_stag, west_east) ;
        MAPFAC_V:FieldType = 104 ;
        MAPFAC_V:MemoryOrder = "XY " ;
        MAPFAC_V:description = "Map scale factor on v-grid" ;
        MAPFAC_V:units = "" ;
        MAPFAC_V:stagger = "Y" ;
        MAPFAC_V:coordinates = "XLONG_V XLAT_V XTIME" ;
float MAPFAC_MX(Time, south_north, west_east) ;
        MAPFAC_MX:FieldType = 104 ;
        MAPFAC_MX:MemoryOrder = "XY " ;
        MAPFAC_MX:description = "Map scale factor on mass grid, x direction" ;
        MAPFAC_MX:units = "" ;
        MAPFAC_MX:stagger = "" ;
        MAPFAC_MX:coordinates = "XLONG XLAT XTIME" ;
float MAPFAC_MY(Time, south_north, west_east) ;
        MAPFAC_MY:FieldType = 104 ;
        MAPFAC_MY:MemoryOrder = "XY " ;
        MAPFAC_MY:description = "Map scale factor on mass grid, y direction" ;
        MAPFAC_MY:units = "" ;
        MAPFAC_MY:stagger = "" ;
        MAPFAC_MY:coordinates = "XLONG XLAT XTIME" ;
float MAPFAC_UX(Time, south_north, west_east_stag) ;
        MAPFAC_UX:FieldType = 104 ;
        MAPFAC_UX:MemoryOrder = "XY " ;
        MAPFAC_UX:description = "Map scale factor on u-grid, x direction" ;
        MAPFAC_UX:units = "" ;
        MAPFAC_UX:stagger = "X" ;
        MAPFAC_UX:coordinates = "XLONG_U XLAT_U XTIME" ;
float MAPFAC_UY(Time, south_north, west_east_stag) ;
        MAPFAC_UY:FieldType = 104 ;
        MAPFAC_UY:MemoryOrder = "XY " ;
        MAPFAC_UY:description = "Map scale factor on u-grid, y direction" ;
        MAPFAC_UY:units = "" ;
        MAPFAC_UY:stagger = "X" ;
        MAPFAC_UY:coordinates = "XLONG_U XLAT_U XTIME" ;
float MAPFAC_VX(Time, south_north_stag, west_east) ;
        MAPFAC_VX:FieldType = 104 ;
        MAPFAC_VX:MemoryOrder = "XY " ;
        MAPFAC_VX:description = "Map scale factor on v-grid, x direction" ;
        MAPFAC_VX:units = "" ;
        MAPFAC_VX:stagger = "Y" ;
        MAPFAC_VX:coordinates = "XLONG_V XLAT_V XTIME" ;
float MF_VX_INV(Time, south_north_stag, west_east) ;
        MF_VX_INV:FieldType = 104 ;
        MF_VX_INV:MemoryOrder = "XY " ;
        MF_VX_INV:description = "Inverse map scale factor on v-grid, x direction" ;
```

```
        MF_VX_INV:units = "" ;
        MF_VX_INV:stagger = "Y" ;
        MF_VX_INV:coordinates = "XLONG_V XLAT_V XTIME" ;
float MAPFAC_VY(Time, south_north_stag, west_east) ;
        MAPFAC_VY:FieldType = 104 ;
        MAPFAC_VY:MemoryOrder = "XY " ;
        MAPFAC_VY:description = "Map scale factor on v-grid, y direction" ;
        MAPFAC_VY:units = "" ;
        MAPFAC_VY:stagger = "Y" ;
        MAPFAC_VY:coordinates = "XLONG_V XLAT_V XTIME" ;
float F(Time, south_north, west_east) ;
        F:FieldType = 104 ;
        F:MemoryOrder = "XY " ;
        F:description = "Coriolis sine latitude term" ;
        F:units = "s-1" ;
        F:stagger = "" ;
        F:coordinates = "XLONG XLAT XTIME" ;
float E(Time, south_north, west_east) ;
        E:FieldType = 104 ;
        E:MemoryOrder = "XY " ;
        E:description = "Coriolis cosine latitude term" ;
        E:units = "s-1" ;
        E:stagger = "" ;
        E:coordinates = "XLONG XLAT XTIME" ;
float SINALPHA(Time, south_north, west_east) ;
        SINALPHA:FieldType = 104 ;
        SINALPHA:MemoryOrder = "XY " ;
        SINALPHA:description = "Local sine of map rotation" ;
        SINALPHA:units = "" ;
        SINALPHA:stagger = "" ;
        SINALPHA:coordinates = "XLONG XLAT XTIME" ;
float COSALPHA(Time, south_north, west_east) ;
        COSALPHA:FieldType = 104 ;
        COSALPHA:MemoryOrder = "XY " ;
        COSALPHA:description = "Local cosine of map rotation" ;
        COSALPHA:units = "" ;
        COSALPHA:stagger = "" ;
        COSALPHA:coordinates = "XLONG XLAT XTIME" ;
float HGT(Time, south_north, west_east) ;
        HGT:FieldType = 104 ;
        HGT:MemoryOrder = "XY " ;
        HGT:description = "Terrain Height" ;
        HGT:units = "m" ;
        HGT:stagger = "" ;
        HGT:coordinates = "XLONG XLAT XTIME" ;
float TSK(Time, south_north, west_east) ;
        TSK:FieldType = 104 ;
        TSK:MemoryOrder = "XY " ;
```

```
        TSK:description = "SURFACE SKIN TEMPERATURE" ;
        TSK:units = "K" ;
        TSK:stagger = "" ;
        TSK:coordinates = "XLONG XLAT XTIME" ;
float P_TOP(Time) ;
        P_TOP:FieldType = 104 ;
        P_TOP:MemoryOrder = "0  " ;
        P_TOP:description = "PRESSURE TOP OF THE MODEL" ;
        P_TOP:units = "Pa" ;
        P_TOP:stagger = "" ;
float T00(Time) ;
        T00:FieldType = 104 ;
        T00:MemoryOrder = "0  " ;
        T00:description = "BASE STATE TEMPERATURE" ;
        T00:units = "K" ;
        T00:stagger = "" ;
float P00(Time) ;
        P00:FieldType = 104 ;
        P00:MemoryOrder = "0  " ;
        P00:description = "BASE STATE PRESURE" ;
        P00:units = "Pa" ;
        P00:stagger = "" ;
float TLP(Time) ;
        TLP:FieldType = 104 ;
        TLP:MemoryOrder = "0  " ;
        TLP:description = "BASE STATE LAPSE RATE" ;
        TLP:units = "" ;
        TLP:stagger = "" ;
float TISO(Time) ;
        TISO:FieldType = 104 ;
        TISO:MemoryOrder = "0  " ;
        TISO:description = "TEMP AT WHICH THE BASE T TURNS CONST" ;
        TISO:units = "K" ;
        TISO:stagger = "" ;
float TLP_STRAT(Time) ;
        TLP_STRAT:FieldType = 104 ;
        TLP_STRAT:MemoryOrder = "0  " ;
        TLP_STRAT:description = "BASE STATE LAPSE RATE (DT/D(LN(P)) IN STRATOSPHERE" ;
        TLP_STRAT:units = "K" ;
        TLP_STRAT:stagger = "" ;
float P_STRAT(Time) ;
        P_STRAT:FieldType = 104 ;
        P_STRAT:MemoryOrder = "0  " ;
        P_STRAT:description = "BASE STATE PRESSURE AT BOTTOM OF STRATOSPHERE" ;
        P_STRAT:units = "Pa" ;
        P_STRAT:stagger = "" ;
float MAX_MSTFX(Time) ;
        MAX_MSTFX:FieldType = 104 ;
```

```
        MAX_MSTFX:MemoryOrder = "0  " ;
        MAX_MSTFX:description = "Max map factor in domain" ;
        MAX_MSTFX:units = "" ;
        MAX_MSTFX:stagger = "" ;
float MAX_MSTFY(Time) ;
        MAX_MSTFY:FieldType = 104 ;
        MAX_MSTFY:MemoryOrder = "0  " ;
        MAX_MSTFY:description = "Max map factor in domain" ;
        MAX_MSTFY:units = "" ;
        MAX_MSTFY:stagger = "" ;
float RAINC(Time, south_north, west_east) ;
        RAINC:FieldType = 104 ;
        RAINC:MemoryOrder = "XY " ;
        RAINC:description = "ACCUMULATED TOTAL CUMULUS PRECIPITATION" ;
        RAINC:units = "mm" ;
        RAINC:stagger = "" ;
        RAINC:coordinates = "XLONG XLAT XTIME" ;
float RAINSH(Time, south_north, west_east) ;
        RAINSH:FieldType = 104 ;
        RAINSH:MemoryOrder = "XY " ;
        RAINSH:description = "ACCUMULATED SHALLOW CUMULUS PRECIPITATION" ;
        RAINSH:units = "mm" ;
        RAINSH:stagger = "" ;
        RAINSH:coordinates = "XLONG XLAT XTIME" ;
float RAINNC(Time, south_north, west_east) ;
        RAINNC:FieldType = 104 ;
        RAINNC:MemoryOrder = "XY " ;
        RAINNC:description = "ACCUMULATED TOTAL GRID SCALE PRECIPITATION" ;
        RAINNC:units = "mm" ;
        RAINNC:stagger = "" ;
        RAINNC:coordinates = "XLONG XLAT XTIME" ;
float SNOWNC(Time, south_north, west_east) ;
        SNOWNC:FieldType = 104 ;
        SNOWNC:MemoryOrder = "XY " ;
        SNOWNC:description = "ACCUMULATED TOTAL GRID SCALE SNOW AND ICE" ;
        SNOWNC:units = "mm" ;
        SNOWNC:stagger = "" ;
        SNOWNC:coordinates = "XLONG XLAT XTIME" ;
float GRAUPELNC(Time, south_north, west_east) ;
        GRAUPELNC:FieldType = 104 ;
        GRAUPELNC:MemoryOrder = "XY " ;
        GRAUPELNC:description = "ACCUMULATED TOTAL GRID SCALE GRAUPEL" ;
        GRAUPELNC:units = "mm" ;
        GRAUPELNC:stagger = "" ;
        GRAUPELNC:coordinates = "XLONG XLAT XTIME" ;
float HAILNC(Time, south_north, west_east) ;
        HAILNC:FieldType = 104 ;
        HAILNC:MemoryOrder = "XY " ;
```

```
        HAILNC:description = "ACCUMULATED TOTAL GRID SCALE HAIL" ;
        HAILNC:units = "mm" ;
        HAILNC:stagger = "" ;
        HAILNC:coordinates = "XLONG XLAT XTIME" ;
float CLDFRA(Time, bottom_top, south_north, west_east) ;
        CLDFRA:FieldType = 104 ;
        CLDFRA:MemoryOrder = "XYZ" ;
        CLDFRA:description = "CLOUD FRACTION" ;
        CLDFRA:units = "" ;
        CLDFRA:stagger = "" ;
        CLDFRA:coordinates = "XLONG XLAT XTIME" ;
float SWDOWN(Time, south_north, west_east) ;
        SWDOWN:FieldType = 104 ;
        SWDOWN:MemoryOrder = "XY " ;
        SWDOWN:description = "DOWNWARD SHORT WAVE FLUX AT GROUND SURFACE" ;
        SWDOWN:units = "W m-2" ;
        SWDOWN:stagger = "" ;
        SWDOWN:coordinates = "XLONG XLAT XTIME" ;
float GLW(Time, south_north, west_east) ;
        GLW:FieldType = 104 ;
        GLW:MemoryOrder = "XY " ;
        GLW:description = "DOWNWARD LONG WAVE FLUX AT GROUND SURFACE" ;
        GLW:units = "W m-2" ;
        GLW:stagger = "" ;
        GLW:coordinates = "XLONG XLAT XTIME" ;
float SWNORM(Time, south_north, west_east) ;
        SWNORM:FieldType = 104 ;
        SWNORM:MemoryOrder = "XY " ;
        SWNORM:description = "NORMAL SHORT WAVE FLUX AT GROUND SURFACE (SLOPE-DEPENDENT
        SWNORM:units = "W m-2" ;
        SWNORM:stagger = "" ;
        SWNORM:coordinates = "XLONG XLAT XTIME" ;
float DIFFUSE_FRAC(Time, south_north, west_east) ;
        DIFFUSE_FRAC:FieldType = 104 ;
        DIFFUSE_FRAC:MemoryOrder = "XY " ;
        DIFFUSE_FRAC:description = "DIFFUSE FRACTION OF SURFACE SHORTWAVE IRRADIANCE"
        DIFFUSE_FRAC:units = "" ;
        DIFFUSE_FRAC:stagger = "" ;
        DIFFUSE_FRAC:coordinates = "XLONG XLAT XTIME" ;
float OLR(Time, south_north, west_east) ;
        OLR:FieldType = 104 ;
        OLR:MemoryOrder = "XY " ;
        OLR:description = "TOA OUTGOING LONG WAVE" ;
        OLR:units = "W m-2" ;
        OLR:stagger = "" ;
        OLR:coordinates = "XLONG XLAT XTIME" ;
float XLAT_U(Time, south_north, west_east_stag) ;
        XLAT_U:FieldType = 104 ;
```

```
        XLAT_U:MemoryOrder = "XY " ;
        XLAT_U:description = "LATITUDE, SOUTH IS NEGATIVE" ;
        XLAT_U:units = "degree_north" ;
        XLAT_U:stagger = "X" ;
        XLAT_U:coordinates = "XLONG_U XLAT_U" ;
float XLONG_U(Time, south_north, west_east_stag) ;
        XLONG_U:FieldType = 104 ;
        XLONG_U:MemoryOrder = "XY " ;
        XLONG_U:description = "LONGITUDE, WEST IS NEGATIVE" ;
        XLONG_U:units = "degree_east" ;
        XLONG_U:stagger = "X" ;
        XLONG_U:coordinates = "XLONG_U XLAT_U" ;
float XLAT_V(Time, south_north_stag, west_east) ;
        XLAT_V:FieldType = 104 ;
        XLAT_V:MemoryOrder = "XY " ;
        XLAT_V:description = "LATITUDE, SOUTH IS NEGATIVE" ;
        XLAT_V:units = "degree_north" ;
        XLAT_V:stagger = "Y" ;
        XLAT_V:coordinates = "XLONG_V XLAT_V" ;
float XLONG_V(Time, south_north_stag, west_east) ;
        XLONG_V:FieldType = 104 ;
        XLONG_V:MemoryOrder = "XY " ;
        XLONG_V:description = "LONGITUDE, WEST IS NEGATIVE" ;
        XLONG_V:units = "degree_east" ;
        XLONG_V:stagger = "Y" ;
        XLONG_V:coordinates = "XLONG_V XLAT_V" ;
float ALBEDO(Time, south_north, west_east) ;
        ALBEDO:FieldType = 104 ;
        ALBEDO:MemoryOrder = "XY " ;
        ALBEDO:description = "ALBEDO" ;
        ALBEDO:units = "-" ;
        ALBEDO:stagger = "" ;
        ALBEDO:coordinates = "XLONG XLAT XTIME" ;
float CLAT(Time, south_north, west_east) ;
        CLAT:FieldType = 104 ;
        CLAT:MemoryOrder = "XY " ;
        CLAT:description = "COMPUTATIONAL GRID LATITUDE, SOUTH IS NEGATIVE" ;
        CLAT:units = "degree_north" ;
        CLAT:stagger = "" ;
        CLAT:coordinates = "XLONG XLAT XTIME" ;
float ALBBCK(Time, south_north, west_east) ;
        ALBBCK:FieldType = 104 ;
        ALBBCK:MemoryOrder = "XY " ;
        ALBBCK:description = "BACKGROUND ALBEDO" ;
        ALBBCK:units = "" ;
        ALBBCK:stagger = "" ;
        ALBBCK:coordinates = "XLONG XLAT XTIME" ;
float EMISS(Time, south_north, west_east) ;
```

```
        EMISS:FieldType = 104 ;
        EMISS:MemoryOrder = "XY " ;
        EMISS:description = "SURFACE EMISSIVITY" ;
        EMISS:units = "" ;
        EMISS:stagger = "" ;
        EMISS:coordinates = "XLONG XLAT XTIME" ;
float NOAHRES(Time, south_north, west_east) ;
        NOAHRES:FieldType = 104 ;
        NOAHRES:MemoryOrder = "XY " ;
        NOAHRES:description = "RESIDUAL OF THE NOAH SURFACE ENERGY BUDGET" ;
        NOAHRES:units = "W m{-2}" ;
        NOAHRES:stagger = "" ;
        NOAHRES:coordinates = "XLONG XLAT XTIME" ;
float TMN(Time, south_north, west_east) ;
        TMN:FieldType = 104 ;
        TMN:MemoryOrder = "XY " ;
        TMN:description = "SOIL TEMPERATURE AT LOWER BOUNDARY" ;
        TMN:units = "K" ;
        TMN:stagger = "" ;
        TMN:coordinates = "XLONG XLAT XTIME" ;
float XLAND(Time, south_north, west_east) ;
        XLAND:FieldType = 104 ;
        XLAND:MemoryOrder = "XY " ;
        XLAND:description = "LAND MASK (1 FOR LAND, 2 FOR WATER)" ;
        XLAND:units = "" ;
        XLAND:stagger = "" ;
        XLAND:coordinates = "XLONG XLAT XTIME" ;
float UST(Time, south_north, west_east) ;
        UST:FieldType = 104 ;
        UST:MemoryOrder = "XY " ;
        UST:description = "U* IN SIMILARITY THEORY" ;
        UST:units = "m s-1" ;
        UST:stagger = "" ;
        UST:coordinates = "XLONG XLAT XTIME" ;
float PBLH(Time, south_north, west_east) ;
        PBLH:FieldType = 104 ;
        PBLH:MemoryOrder = "XY " ;
        PBLH:description = "PBL HEIGHT" ;
        PBLH:units = "m" ;
        PBLH:stagger = "" ;
        PBLH:coordinates = "XLONG XLAT XTIME" ;
float HFX(Time, south_north, west_east) ;
        HFX:FieldType = 104 ;
        HFX:MemoryOrder = "XY " ;
        HFX:description = "UPWARD HEAT FLUX AT THE SURFACE" ;
        HFX:units = "W m-2" ;
        HFX:stagger = "" ;
        HFX:coordinates = "XLONG XLAT XTIME" ;
```

```
float QFX(Time, south_north, west_east) ;
        QFX:FieldType = 104 ;
        QFX:MemoryOrder = "XY " ;
        QFX:description = "UPWARD MOISTURE FLUX AT THE SURFACE" ;
        QFX:units = "kg m-2 s-1" ;
        QFX:stagger = "" ;
        QFX:coordinates = "XLONG XLAT XTIME" ;
float LH(Time, south_north, west_east) ;
        LH:FieldType = 104 ;
        LH:MemoryOrder = "XY " ;
        LH:description = "LATENT HEAT FLUX AT THE SURFACE" ;
        LH:units = "W m-2" ;
        LH:stagger = "" ;
        LH:coordinates = "XLONG XLAT XTIME" ;
float ACHFX(Time, south_north, west_east) ;
        ACHFX:FieldType = 104 ;
        ACHFX:MemoryOrder = "XY " ;
        ACHFX:description = "ACCUMULATED UPWARD HEAT FLUX AT THE SURFACE" ;
        ACHFX:units = "J m-2" ;
        ACHFX:stagger = "" ;
        ACHFX:coordinates = "XLONG XLAT XTIME" ;
float ACLHF(Time, south_north, west_east) ;
        ACLHF:FieldType = 104 ;
        ACLHF:MemoryOrder = "XY " ;
        ACLHF:description = "ACCUMULATED UPWARD LATENT HEAT FLUX AT THE SURFACE" ;
        ACLHF:units = "J m-2" ;
        ACLHF:stagger = "" ;
        ACLHF:coordinates = "XLONG XLAT XTIME" ;
float SNOWC(Time, south_north, west_east) ;
        SNOWC:FieldType = 104 ;
        SNOWC:MemoryOrder = "XY " ;
        SNOWC:description = "FLAG INDICATING SNOW COVERAGE (1 FOR SNOW COVER)" ;
        SNOWC:units = "" ;
        SNOWC:stagger = "" ;
        SNOWC:coordinates = "XLONG XLAT XTIME" ;
float SR(Time, south_north, west_east) ;
        SR:FieldType = 104 ;
        SR:MemoryOrder = "XY " ;
        SR:description = "fraction of frozen precipitation" ;
        SR:units = "-" ;
        SR:stagger = "" ;
        SR:coordinates = "XLONG XLAT XTIME" ;
int SAVE_TOPO_FROM_REAL(Time) ;
        SAVE_TOPO_FROM_REAL:FieldType = 106 ;
        SAVE_TOPO_FROM_REAL:MemoryOrder = "0  " ;
        SAVE_TOPO_FROM_REAL:description = "1=original topo from real/0=topo modified by
        SAVE_TOPO_FROM_REAL:units = "flag" ;
        SAVE_TOPO_FROM_REAL:stagger = "" ;
```

```
        int ISEEDARR_RAND_PERTURB(Time, bottom_top) ;
                ISEEDARR_RAND_PERTURB:FieldType = 106 ;
                ISEEDARR_RAND_PERTURB:MemoryOrder = "Z  " ;
                ISEEDARR_RAND_PERTURB:description = "Array to hold seed for restart, RAND_PERT
                ISEEDARR_RAND_PERTURB:units = "" ;
                ISEEDARR_RAND_PERTURB:stagger = "" ;
        int ISEEDARR_SPPT(Time, bottom_top) ;
                ISEEDARR_SPPT:FieldType = 106 ;
                ISEEDARR_SPPT:MemoryOrder = "Z  " ;
                ISEEDARR_SPPT:description = "Array to hold seed for restart, SPPT" ;
                ISEEDARR_SPPT:units = "" ;
                ISEEDARR_SPPT:stagger = "" ;
        int ISEEDARR_SKEBS(Time, bottom_top) ;
                ISEEDARR_SKEBS:FieldType = 106 ;
                ISEEDARR_SKEBS:MemoryOrder = "Z  " ;
                ISEEDARR_SKEBS:description = "Array to hold seed for restart, SKEBS" ;
                ISEEDARR_SKEBS:units = "" ;
                ISEEDARR_SKEBS:stagger = "" ;
        float LANDMASK(Time, south_north, west_east) ;
                LANDMASK:FieldType = 104 ;
                LANDMASK:MemoryOrder = "XY " ;
                LANDMASK:description = "LAND MASK (1 FOR LAND, 0 FOR WATER)" ;
                LANDMASK:units = "" ;
                LANDMASK:stagger = "" ;
                LANDMASK:coordinates = "XLONG XLAT XTIME" ;
        float LAKEMASK(Time, south_north, west_east) ;
                LAKEMASK:FieldType = 104 ;
                LAKEMASK:MemoryOrder = "XY " ;
                LAKEMASK:description = "LAKE MASK (1 FOR LAKE, 0 FOR NON-LAKE)" ;
                LAKEMASK:units = "" ;
                LAKEMASK:stagger = "" ;
                LAKEMASK:coordinates = "XLONG XLAT XTIME" ;
        float SST(Time, south_north, west_east) ;
                SST:FieldType = 104 ;
                SST:MemoryOrder = "XY " ;
                SST:description = "SEA SURFACE TEMPERATURE" ;
                SST:units = "K" ;
                SST:stagger = "" ;
                SST:coordinates = "XLONG XLAT XTIME" ;
        float SST_INPUT(Time, south_north, west_east) ;
                SST_INPUT:FieldType = 104 ;
                SST_INPUT:MemoryOrder = "XY " ;
                SST_INPUT:description = "SEA SURFACE TEMPERATURE FROM WRFLOWINPUT FILE" ;
                SST_INPUT:units = "K" ;
                SST_INPUT:stagger = "" ;
                SST_INPUT:coordinates = "XLONG XLAT XTIME" ;

// global attributes:
```

```
:TITLE = " OUTPUT FROM WRF V3.7 MODEL" ;
:START_DATE = "2005-08-28_00:00:00" ;
:SIMULATION_START_DATE = "2005-08-28_00:00:00" ;
:WEST-EAST_GRID_DIMENSION = 91 ;
:SOUTH-NORTH_GRID_DIMENSION = 74 ;
:BOTTOM-TOP_GRID_DIMENSION = 30 ;
:DX = 30000.f ;
:DY = 30000.f ;
:SKEBS_ON = 0 ;
:SPEC_BDY_FINAL_MU = 0 ;
:USE_Q_DIABATIC = 0 ;
:GRIDTYPE = "C" ;
:DIFF_OPT = 1 ;
:KM_OPT = 4 ;
:DAMP_OPT = 0 ;
:DAMPCOEF = 0.2f ;
:KHDIF = 0.f ;
:KVDIF = 0.f ;
:MP_PHYSICS = 3 ;
:RA_LW_PHYSICS = 1 ;
:RA_SW_PHYSICS = 1 ;
:SF_SFCLAY_PHYSICS = 1 ;
:SF_SURFACE_PHYSICS = 2 ;
:BL_PBL_PHYSICS = 1 ;
:CU_PHYSICS = 1 ;
:SF_LAKE_PHYSICS = 0 ;
:SURFACE_INPUT_SOURCE = 3 ;
:SST_UPDATE = 0 ;
:GRID_FDDA = 0 ;
:GFDDA_INTERVAL_M = 0 ;
:GFDDA_END_H = 0 ;
:GRID_SFDDA = 0 ;
:SGFDDA_INTERVAL_M = 0 ;
:SGFDDA_END_H = 0 ;
:HYPSOMETRIC_OPT = 2 ;
:USE_THETA_M = 0 ;
:SF_URBAN_PHYSICS = 0 ;
:SHCU_PHYSICS = 0 ;
:MFSHCONV = 0 ;
:FEEDBACK = 1 ;
:SMOOTH_OPTION = 0 ;
:SWRAD_SCAT = 1.f ;
:W_DAMPING = 0 ;
:DT = 180.f ;
:RADT = 30.f ;
:BLDT = 0.f ;
:CUDT = 5.f ;
:AER_OPT = 0 ;
```

```
:SWINT_OPT = 0 ;
:AER_TYPE = 1 ;
:AER_AOD550_OPT = 1 ;
:AER_ANGEXP_OPT = 1 ;
:AER_SSA_OPT = 1 ;
:AER_ASY_OPT = 1 ;
:AER_AOD550_VAL = 0.12f ;
:AER_ANGEXP_VAL = 1.3f ;
:AER_SSA_VAL = 1.401298e-45f ;
:AER_ASY_VAL = 1.401298e-45f ;
:MOIST_ADV_OPT = 1 ;
:SCALAR_ADV_OPT = 1 ;
:TKE_ADV_OPT = 1 ;
:DIFF_6TH_OPT = 0 ;
:DIFF_6TH_FACTOR = 0.12f ;
:OBS_NUDGE_OPT = 0 ;
:BUCKET_MM = -1.f ;
:BUCKET_J = -1.f ;
:PREC_ACC_DT = 0.f ;
:SF_OCEAN_PHYSICS = 0 ;
:ISFTCFLX = 0 ;
:ISHALLOW = 0 ;
:ISFFLX = 1 ;
:ICLOUD = 1 ;
:ICLOUD_CU = 0 ;
:TRACER_PBLMIX = 1 ;
:SCALAR_PBLMIX = 0 ;
:YSU_TOPDOWN_PBLMIX = 0 ;
:GRAV_SETTLING = 0 ;
:DFI_OPT = 0 ;
:SIMULATION_INITIALIZATION_TYPE = "REAL-DATA CASE" ;
:WEST-EAST_PATCH_START_UNSTAG = 1 ;
:WEST-EAST_PATCH_END_UNSTAG = 90 ;
:WEST-EAST_PATCH_START_STAG = 1 ;
:WEST-EAST_PATCH_END_STAG = 91 ;
:SOUTH-NORTH_PATCH_START_UNSTAG = 1 ;
:SOUTH-NORTH_PATCH_END_UNSTAG = 73 ;
:SOUTH-NORTH_PATCH_START_STAG = 1 ;
:SOUTH-NORTH_PATCH_END_STAG = 74 ;
:BOTTOM-TOP_PATCH_START_UNSTAG = 1 ;
:BOTTOM-TOP_PATCH_END_UNSTAG = 29 ;
:BOTTOM-TOP_PATCH_START_STAG = 1 ;
:BOTTOM-TOP_PATCH_END_STAG = 30 ;
:GRID_ID = 1 ;
:PARENT_ID = 0 ;
:I_PARENT_START = 1 ;
:J_PARENT_START = 1 ;
:PARENT_GRID_RATIO = 1 ;
```

```
                    :CEN_LAT = 27.99999f ;
                    :CEN_LON = -89.f ;
                    :TRUELAT1 = 0.f ;
                    :TRUELAT2 = 0.f ;
                    :MOAD_CEN_LAT = 27.99999f ;
                    :STAND_LON = -89.f ;
                    :POLE_LAT = 90.f ;
                    :POLE_LON = 0.f ;
                    :GMT = 0.f ;
                    :JULYR = 2005 ;
                    :JULDAY = 240 ;
                    :MAP_PROJ = 3 ;
                    :MAP_PROJ_CHAR = "Mercator" ;
                    :MMINLU = "USGS" ;
                    :NUM_LAND_CAT = 24 ;
                    :ISWATER = 16 ;
                    :ISLAKE = -1 ;
                    :ISICE = 24 ;
                    :ISURBAN = 1 ;
                    :ISOILWATER = 14 ;
}
```

12.11 Reading a WRF File in Python

You have several options to read a WRF NetCDF file in Python.

- **netcdf4-python**
- PyNIO (currently Python 2.x only)
- xarray (Dataset not currently supported in wrf-python)

12.12 netcdf4-python Example

```
from netCDF4 import Dataset

file_path = "./wrfout_d01_2005-08-28_00:00:00"

wrf_file = Dataset(file_path)
```

12.13 Your Turn!

12.13.1 Example 2.2: Using netcdf4-python

```
In [6]: from netCDF4 import Dataset

        file_path = single_wrf_file()

        wrf_file = Dataset(file_path)
```

```
        print(wrf_file)
```

```
<type 'netCDF4._netCDF4.Dataset'>
root group (NETCDF3_CLASSIC data model, file format NETCDF3):
    TITLE:  OUTPUT FROM WRF V3.7 MODEL
    START_DATE: 2005-08-28_00:00:00
    SIMULATION_START_DATE: 2005-08-28_00:00:00
    WEST-EAST_GRID_DIMENSION: 91
    SOUTH-NORTH_GRID_DIMENSION: 74
    BOTTOM-TOP_GRID_DIMENSION: 30
    DX: 30000.0
    DY: 30000.0
    SKEBS_ON: 0
    SPEC_BDY_FINAL_MU: 0
    USE_Q_DIABATIC: 0
    GRIDTYPE: C
    DIFF_OPT: 1
    KM_OPT: 4
    DAMP_OPT: 0
    DAMPCOEF: 0.2
    KHDIF: 0.0
    KVDIF: 0.0
    MP_PHYSICS: 3
    RA_LW_PHYSICS: 1
    RA_SW_PHYSICS: 1
    SF_SFCLAY_PHYSICS: 1
    SF_SURFACE_PHYSICS: 2
    BL_PBL_PHYSICS: 1
    CU_PHYSICS: 1
    SF_LAKE_PHYSICS: 0
    SURFACE_INPUT_SOURCE: 3
    SST_UPDATE: 0
    GRID_FDDA: 0
    GFDDA_INTERVAL_M: 0
    GFDDA_END_H: 0
    GRID_SFDDA: 0
    SGFDDA_INTERVAL_M: 0
    SGFDDA_END_H: 0
    HYPSOMETRIC_OPT: 2
    USE_THETA_M: 0
    SF_URBAN_PHYSICS: 0
    SHCU_PHYSICS: 0
    MFSHCONV: 0
    FEEDBACK: 1
    SMOOTH_OPTION: 0
    SWRAD_SCAT: 1.0
    W_DAMPING: 0
```

```
DT: 180.0
RADT: 30.0
BLDT: 0.0
CUDT: 5.0
AER_OPT: 0
SWINT_OPT: 0
AER_TYPE: 1
AER_AOD550_OPT: 1
AER_ANGEXP_OPT: 1
AER_SSA_OPT: 1
AER_ASY_OPT: 1
AER_AOD550_VAL: 0.12
AER_ANGEXP_VAL: 1.3
AER_SSA_VAL: 1.4013e-45
AER_ASY_VAL: 1.4013e-45
MOIST_ADV_OPT: 1
SCALAR_ADV_OPT: 1
TKE_ADV_OPT: 1
DIFF_6TH_OPT: 0
DIFF_6TH_FACTOR: 0.12
OBS_NUDGE_OPT: 0
BUCKET_MM: -1.0
BUCKET_J: -1.0
PREC_ACC_DT: 0.0
SF_OCEAN_PHYSICS: 0
ISFTCFLX: 0
ISHALLOW: 0
ISFFLX: 1
ICLOUD: 1
ICLOUD_CU: 0
TRACER_PBLMIX: 1
SCALAR_PBLMIX: 0
YSU_TOPDOWN_PBLMIX: 0
GRAV_SETTLING: 0
DFI_OPT: 0
SIMULATION_INITIALIZATION_TYPE: REAL-DATA CASE
WEST-EAST_PATCH_START_UNSTAG: 1
WEST-EAST_PATCH_END_UNSTAG: 90
WEST-EAST_PATCH_START_STAG: 1
WEST-EAST_PATCH_END_STAG: 91
SOUTH-NORTH_PATCH_START_UNSTAG: 1
SOUTH-NORTH_PATCH_END_UNSTAG: 73
SOUTH-NORTH_PATCH_START_STAG: 1
SOUTH-NORTH_PATCH_END_STAG: 74
BOTTOM-TOP_PATCH_START_UNSTAG: 1
BOTTOM-TOP_PATCH_END_UNSTAG: 29
BOTTOM-TOP_PATCH_START_STAG: 1
BOTTOM-TOP_PATCH_END_STAG: 30
```

```
GRID_ID: 1
PARENT_ID: 0
I_PARENT_START: 1
J_PARENT_START: 1
PARENT_GRID_RATIO: 1
CEN_LAT: 28.0
CEN_LON: -89.0
TRUELAT1: 0.0
TRUELAT2: 0.0
MOAD_CEN_LAT: 28.0
STAND_LON: -89.0
POLE_LAT: 90.0
POLE_LON: 0.0
GMT: 0.0
JULYR: 2005
JULDAY: 240
MAP_PROJ: 3
MAP_PROJ_CHAR: Mercator
MMINLU: USGS
NUM_LAND_CAT: 24
ISWATER: 16
ISLAKE: -1
ISICE: 24
ISURBAN: 1
ISOILWATER: 14
dimensions(sizes): Time(4), DateStrLen(19), west_east(90), south_north(73), bottom_top(29)
variables(dimensions): |S1 Times(Time,DateStrLen), float32 XLAT(Time,south_north,west_east)
groups:
```

12.14 Getting Variables and Attributes

- netcdf4-python uses an old API that was originally created for a package called Scientific.IO.NetCDF.
- PyNIO also uses this API.
- xarray does not use this API.
- The API may look a little dated.

12.14.1 Getting global attributes

The get the full dictionary of global attributes, use the __dict__ attribute.

To work with one attribute at a time, you can use the getncattr and setncattr methods.

```
global_attrs = wrf_file.__dict__
```

```
# To get the value for MAP_PROJ, you can do:
map_proj = wrf_file.__dict__["MAP_PROJ"]

# Or more cleanly
map_prof = wrf_file.getncattr("MAP_PROJ")
```

12.14.2 Getting variables, variable attributes, and variable data

All variables are stored in a dictionary attribute called *variables*.

Let's get the perturbation pressure ("P") variable.

```
# This will return a netCDF4.Variable object
p = wrf_file.variables["P"]
```

To get the variable attributes, you can use the __dict__ attribute to get a dictionary of all attributes.

Use the *getncattr* function if you already know the attribute name.

```
# Return a dictionary of all of P's
# attributes
p_attrs = p.__dict__

# Let's just get the 'coordinates' attribute
p_coords = p.getncattr("coordinates")
```

To get the variable's data as a numpy array, you need to use Python's [ ] API (__getitem__ for those that are more familiar with Python's data model).

```
# Get a numpy array for all times
p_all_data = p[:,:,:,:]

# A shorthand version of the above.
p_all_data = p[:]

# This will extract the numpy array for
# time index 0.

p_t0_data = p[0,:]
```

12.15 Your Turn!

12.15.1 Example 2.3: Variables, Attributes, and Data with netcd4-python

```
In [7]: from netCDF4 import Dataset

        file_path = single_wrf_file()
```

```
        # Create the netCDF4.Dataset object
        wrf_file = Dataset(file_path)

        # Get the global attribute dict
        global_attrs = wrf_file.__dict__
        print ("Global attributes for the file")
        print(global_attrs)
        print ("\n")

        # Just get the 'MAP_PROJ' attribute
        map_proj = wrf_file.getncattr("MAP_PROJ")
        print ("The MAP_PROJ attribute:")
        print (map_proj)
        print("\n")

        # Get the perturbation pressure variable
        p = wrf_file.variables["P"]
        print ("The P variable: ")
        print(p)
        print ("\n")

        # Get the P attributes
        p_attrs = p.__dict__
        print ("The attribute dict for P")
        print (p_attrs)
        print ("\n")

        # Get the 'coordinates' attribute for P
        coords = p.getncattr("coordinates")
        print ("Coordinates for P:")
        print (coords)
        print ("\n")

        # Get the P numpy array for all times
        p_all_data = p[:]
        print ("The P numpy array: ")
        print (p_all_data)
        print ("\n")

        # Get the P numpy array for time 0
        p_t0_data = p[0,:]
        print ("P array at time 0:")
        print (p_t0_data)
        print ("\n")
```

```
Global attributes for the file
OrderedDict([(u'TITLE', u' OUTPUT FROM WRF V3.7 MODEL'), (u'START_DATE', u'2005-08-28_00:00:00
```

```
The MAP_PROJ attribute:
3


The P variable:
<type 'netCDF4._netCDF4.Variable'>
float32 P(Time, bottom_top, south_north, west_east)
    FieldType: 104
    MemoryOrder: XYZ
    description: perturbation pressure
    units: Pa
    stagger:
    coordinates: XLONG XLAT XTIME
unlimited dimensions: Time
current shape = (4, 29, 73, 90)
filling off



The attribute dict for P
OrderedDict([(u'FieldType', 104), (u'MemoryOrder', u'XYZ'), (u'description', u'perturbation pr


Coordinates for P:
XLONG XLAT XTIME


The P numpy array:
[[[[  9.74273438e+02   1.00946875e+03   1.18368750e+03 ...,
      8.73070312e+02   8.83312500e+02   8.94546875e+02]
   [  1.22443750e+03   1.24510938e+03   1.35247656e+03 ...,
      8.93570312e+02   9.17164062e+02   9.84453125e+02]
   [  1.39332812e+03   1.44563281e+03   1.51516406e+03 ...,
      9.52703125e+02   1.05688281e+03   1.04038281e+03]
   ...,
   [  1.26719531e+03   1.27103906e+03   1.26158594e+03 ...,
      1.33459375e+03   1.33853125e+03   1.34417969e+03]
   [  1.24639844e+03   1.24543750e+03   1.24398438e+03 ...,
      1.34735156e+03   1.35038281e+03   1.35546094e+03]
   [  1.24002344e+03   1.23860938e+03   1.23604688e+03 ...,
      1.36002344e+03   1.36113281e+03   1.36603125e+03]]

  [[  9.54585938e+02   9.91156250e+02   1.16360938e+03 ...,
      8.54710938e+02   8.64906250e+02   8.76039062e+02]
   [  1.20507031e+03   1.22354688e+03   1.33133594e+03 ...,
      8.74679688e+02   8.88554688e+02   9.04328125e+02]
```

```
 [  1.37400781e+03   1.42587500e+03   1.49564062e+03 ...,
    9.02445312e+02   9.21664062e+02   9.38984375e+02]
 ...,
 [  1.24985156e+03   1.25361719e+03   1.24376562e+03 ...,
    1.31405469e+03   1.31810938e+03   1.32390625e+03]
 [  1.22927344e+03   1.22803125e+03   1.22610156e+03 ...,
    1.32667969e+03   1.33000000e+03   1.33527344e+03]
 [  1.22284375e+03   1.22117969e+03   1.21827344e+03 ...,
    1.33940625e+03   1.34070312e+03   1.34578125e+03]]

[[  9.29390625e+02   9.66953125e+02   1.13655469e+03 ...,
    8.30562500e+02   8.40585938e+02   8.51625000e+02]
 [  1.17989062e+03   1.19630469e+03   1.30484375e+03 ...,
    8.50187500e+02   8.63789062e+02   8.78562500e+02]
 [  1.34799219e+03   1.39972656e+03   1.47011719e+03 ...,
    8.76710938e+02   8.96023438e+02   9.12781250e+02]
 ...,
 [  1.22655469e+03   1.22996875e+03   1.22019531e+03 ...,
    1.28667188e+03   1.29094531e+03   1.29697656e+03]
 [  1.20612500e+03   1.20460938e+03   1.20247656e+03 ...,
    1.29932812e+03   1.30290625e+03   1.30835938e+03]
 [  1.19992188e+03   1.19785156e+03   1.19479688e+03 ...,
    1.31199219e+03   1.31358594e+03   1.31896094e+03]]

...,
[[  1.52529297e+01   1.49389648e+01   1.94331055e+01 ...,
    9.24658203e+00   9.51513672e+00   9.71826172e+00]
 [  2.39931641e+01   2.36577148e+01   2.69301758e+01 ...,
    9.61279297e+00   9.86816406e+00   1.01264648e+01]
 [  2.98544922e+01   3.14418945e+01   3.35732422e+01 ...,
    1.01445312e+01   1.04985352e+01   1.07553711e+01]
 ...,
 [  2.67846680e+01   2.65302734e+01   2.57548828e+01 ...,
    2.23295898e+01   2.22187500e+01   2.22534180e+01]
 [  2.64399414e+01   2.60390625e+01   2.55820312e+01 ...,
    2.29863281e+01   2.28554688e+01   2.28803711e+01]
 [  2.65966797e+01   2.61943359e+01   2.57241211e+01 ...,
    2.35556641e+01   2.34252930e+01   2.34877930e+01]]

[[  8.44482422e+00   8.27246094e+00   1.07617188e+01 ...,
    5.12695312e+00   5.27636719e+00   5.37158203e+00]
 [  1.33017578e+01   1.31030273e+01   1.49233398e+01 ...,
    5.31494141e+00   5.46923828e+00   5.61376953e+00]
 [  1.65483398e+01   1.74174805e+01   1.86010742e+01 ...,
    5.61279297e+00   5.81933594e+00   5.96337891e+00]
 ...,
 [  1.48398438e+01   1.47031250e+01   1.42675781e+01 ...,
    1.23769531e+01   1.23027344e+01   1.23295898e+01]
```

```
 [  1.46474609e+01   1.44277344e+01   1.41772461e+01 ...,
    1.27363281e+01   1.26665039e+01   1.26796875e+01]
 [  1.47412109e+01   1.45068359e+01   1.42421875e+01 ...,
    1.30561523e+01   1.29716797e+01   1.30107422e+01]]

[[  2.65966797e+00   2.60253906e+00   3.39355469e+00 ...,
    1.61523438e+00   1.65576172e+00   1.69189453e+00]
 [  4.18066406e+00   4.13378906e+00   4.70117188e+00 ...,
    1.67089844e+00   1.72070312e+00   1.76171875e+00]
 [  5.20361328e+00   5.49218750e+00   5.85595703e+00 ...,
    1.77587891e+00   1.83496094e+00   1.87158203e+00]
 ...,
 [  4.66845703e+00   4.62988281e+00   4.49121094e+00 ...,
    3.89599609e+00   3.87841797e+00   3.87890625e+00]
 [  4.62060547e+00   4.54931641e+00   4.46826172e+00 ...,
    4.00585938e+00   3.98095703e+00   3.99169922e+00]
 [  4.63720703e+00   4.57568359e+00   4.48974609e+00 ...,
    4.11523438e+00   4.08496094e+00   4.09130859e+00]]]


[[[  1.05386719e+03   1.11839844e+03   1.27633594e+03 ...,
    9.01546875e+02   9.09007812e+02   9.18000000e+02]
 [  1.27593750e+03   1.30658594e+03   1.42550781e+03 ...,
    9.16867188e+02   9.31335938e+02   1.00160156e+03]
 [  1.42446094e+03   1.51921094e+03   1.59873438e+03 ...,
    9.31078125e+02   9.54398438e+02   1.05418750e+03]
 ...,
 [  1.26764062e+03   1.25415625e+03   1.23252344e+03 ...,
    1.23791406e+03   1.24307031e+03   1.24384375e+03]
 [  1.25731250e+03   1.23817188e+03   1.22482812e+03 ...,
    1.25071875e+03   1.25204688e+03   1.25823438e+03]
 [  1.25689844e+03   1.24856250e+03   1.23771875e+03 ...,
    1.27219531e+03   1.26853125e+03   1.26967969e+03]]

[[  1.03731250e+03   1.10100781e+03   1.25506250e+03 ...,
    8.84070312e+02   8.91882812e+02   8.99953125e+02]
 [  1.25553125e+03   1.28857031e+03   1.40796875e+03 ...,
    8.98539062e+02   9.13531250e+02   9.23562500e+02]
 [  1.40610156e+03   1.49926562e+03   1.58189844e+03 ...,
    9.12070312e+02   9.37281250e+02   9.52710938e+02]
 ...,
 [  1.24816406e+03   1.23818750e+03   1.21635938e+03 ...,
    1.21915625e+03   1.22095312e+03   1.22577344e+03]
 [  1.23886719e+03   1.22330469e+03   1.20830469e+03 ...,
    1.23350781e+03   1.23726562e+03   1.23994531e+03]
 [  1.23844531e+03   1.23164062e+03   1.22016406e+03 ...,
    1.25284375e+03   1.25064062e+03   1.25185156e+03]]
```

```
[[  1.01671875e+03   1.07336719e+03   1.22559375e+03 ...,
    8.59671875e+02   8.67578125e+02   8.75781250e+02]
 [  1.23352344e+03   1.26232031e+03   1.38342188e+03 ...,
    8.75242188e+02   8.87828125e+02   8.98468750e+02]
 [  1.38213281e+03   1.47528906e+03   1.55543750e+03 ...,
    8.86906250e+02   9.10609375e+02   9.26437500e+02]
 ...,
 [  1.22613281e+03   1.21397656e+03   1.19151562e+03 ...,
    1.19490625e+03   1.19669531e+03   1.19934375e+03]
 [  1.21490625e+03   1.19924219e+03   1.18456250e+03 ...,
    1.20679688e+03   1.20975781e+03   1.21281250e+03]
 [  1.21581250e+03   1.20685156e+03   1.19583594e+03 ...,
    1.22835938e+03   1.22560156e+03   1.22693750e+03]]


...,
[[  1.71435547e+01   1.74418945e+01   2.17226562e+01 ...,
    9.84716797e+00   1.00488281e+01   1.01826172e+01]
 [  2.50517578e+01   2.53110352e+01   2.89775391e+01 ...,
    1.04125977e+01   1.06474609e+01   1.04775391e+01]
 [  3.06264648e+01   3.33984375e+01   3.58891602e+01 ...,
    1.06337891e+01   1.11044922e+01   1.09775391e+01]
 ...,
 [  2.68359375e+01   2.66723633e+01   2.57661133e+01 ...,
    2.00751953e+01   1.98779297e+01   1.95595703e+01]
 [  2.66586914e+01   2.62426758e+01   2.55947266e+01 ...,
    2.07192383e+01   2.05419922e+01   2.01630859e+01]
 [  2.68388672e+01   2.64028320e+01   2.58500977e+01 ...,
    2.10971680e+01   2.08647461e+01   2.08002930e+01]]

[[  9.52441406e+00   9.68261719e+00   1.20527344e+01 ...,
    5.44580078e+00   5.55810547e+00   5.62841797e+00]
 [  1.39008789e+01   1.40000000e+01   1.60346680e+01 ...,
    5.73046875e+00   5.86035156e+00   5.79638672e+00]
 [  1.69902344e+01   1.84702148e+01   1.98623047e+01 ...,
    5.84179688e+00   6.10546875e+00   6.09667969e+00]
 ...,
 [  1.48837891e+01   1.47739258e+01   1.42807617e+01 ...,
    1.11123047e+01   1.10112305e+01   1.07954102e+01]
 [  1.47802734e+01   1.45351562e+01   1.41831055e+01 ...,
    1.14750977e+01   1.13759766e+01   1.11323242e+01]
 [  1.48720703e+01   1.46230469e+01   1.43090820e+01 ...,
    1.16621094e+01   1.15185547e+01   1.14892578e+01]]

[[  3.02978516e+00   3.09326172e+00   3.82958984e+00 ...,
    1.72021484e+00   1.74902344e+00   1.77246094e+00]
 [  4.38720703e+00   4.34765625e+00   4.99560547e+00 ...,
    1.73095703e+00   1.77246094e+00   1.82617188e+00]
 [  5.35498047e+00   5.75341797e+00   6.19873047e+00 ...,
```

```
    1.76660156e+00   1.85107422e+00   1.91992188e+00]
  ...,
  [  4.68359375e+00   4.64062500e+00   4.48291016e+00 ...,
     3.49462891e+00   3.45507812e+00   3.33789062e+00]
  [  4.66357422e+00   4.55908203e+00   4.45507812e+00 ...,
     3.60302734e+00   3.56152344e+00   3.44921875e+00]
  [  4.69140625e+00   4.62207031e+00   4.51855469e+00 ...,
     3.61425781e+00   3.57470703e+00   3.54833984e+00]]]


[[[  1.14628125e+03   1.22757812e+03   1.36558594e+03 ...,
     9.30132812e+02   9.37414062e+02   9.44210938e+02]
  [  1.33117188e+03   1.37914844e+03   1.48456250e+03 ...,
     9.39546875e+02   9.52851562e+02   1.01887500e+03]
  [  1.46105469e+03   1.54947656e+03   1.63157031e+03 ...,
     9.45500000e+02   9.68632812e+02   1.07066406e+03]
  ...,
  [  1.26528125e+03   1.25151562e+03   1.22432031e+03 ...,
     1.14467969e+03   1.14360156e+03   1.14711719e+03]
  [  1.26533594e+03   1.24628906e+03   1.22534375e+03 ...,
     1.16169531e+03   1.16084375e+03   1.16248438e+03]
  [  1.27103906e+03   1.25679688e+03   1.23897656e+03 ...,
     1.18653125e+03   1.18035156e+03   1.17745312e+03]]

 [[  1.12939062e+03   1.20731250e+03   1.34388281e+03 ...,
     9.11789062e+02   9.19070312e+02   9.25835938e+02]
  [  1.31341406e+03   1.36164062e+03   1.46564844e+03 ...,
     9.18203125e+02   9.34039062e+02   9.42843750e+02]
  [  1.44284375e+03   1.52958594e+03   1.61099219e+03 ...,
     9.25867188e+02   9.48835938e+02   9.64875000e+02]
  ...,
  [  1.24725781e+03   1.23256250e+03   1.20546875e+03 ...,
     1.12639844e+03   1.12586719e+03   1.12867188e+03]
  [  1.24713281e+03   1.22899219e+03   1.20796094e+03 ...,
     1.14479688e+03   1.14307812e+03   1.14417188e+03]
  [  1.25299219e+03   1.23853125e+03   1.22109375e+03 ...,
     1.16825000e+03   1.16213281e+03   1.15921875e+03]]

 [[  1.10717969e+03   1.18040625e+03   1.31479688e+03 ...,
     8.87570312e+02   8.94734375e+02   9.01406250e+02]
  [  1.28930469e+03   1.33740625e+03   1.44201562e+03 ...,
     8.93187500e+02   9.09476562e+02   9.17281250e+02]
  [  1.41832031e+03   1.50538281e+03   1.58673438e+03 ...,
     9.01265625e+02   9.24289062e+02   9.40265625e+02]
  ...,
  [  1.22256250e+03   1.20910156e+03   1.18175000e+03 ...,
     1.10319531e+03   1.10261719e+03   1.10385938e+03]
  [  1.22247656e+03   1.20428906e+03   1.18523438e+03 ...,
```

```
    1.12027344e+03   1.11964062e+03   1.11957812e+03]
 [  1.22829688e+03   1.21394531e+03   1.19655469e+03 ...,
    1.14364062e+03   1.13760156e+03   1.13464062e+03]]

...,
[[  1.89604492e+01   1.98041992e+01   2.38759766e+01 ...,
    1.04404297e+01   1.05756836e+01   1.06386719e+01]
 [  2.60522461e+01   2.71552734e+01   3.06127930e+01 ...,
    1.07294922e+01   1.09902344e+01   1.07958984e+01]
 [  3.13603516e+01   3.42543945e+01   3.70317383e+01 ...,
    1.08095703e+01   1.12719727e+01   1.11958008e+01]
 ...,
 [  2.69169922e+01   2.66044922e+01   2.57153320e+01 ...,
    1.75590820e+01   1.73354492e+01   1.69555664e+01]
 [  2.68862305e+01   2.63750000e+01   2.56967773e+01 ...,
    1.82885742e+01   1.80146484e+01   1.75522461e+01]
 [  2.70839844e+01   2.66015625e+01   2.59619141e+01 ...,
    1.87260742e+01   1.83999023e+01   1.82290039e+01]]

[[  1.05302734e+01   1.10112305e+01   1.32651367e+01 ...,
    5.78955078e+00   5.84179688e+00   5.89697266e+00]
 [  1.44384766e+01   1.50205078e+01   1.69291992e+01 ...,
    5.90478516e+00   6.04736328e+00   5.98144531e+00]
 [  1.73989258e+01   1.89580078e+01   2.04980469e+01 ...,
    5.95068359e+00   6.21582031e+00   6.20947266e+00]
 ...,
 [  1.49223633e+01   1.47426758e+01   1.42539062e+01 ...,
    9.72802734e+00   9.59130859e+00   9.34179688e+00]
 [  1.49008789e+01   1.45986328e+01   1.42324219e+01 ...,
    1.01176758e+01   9.98095703e+00   9.68164062e+00]
 [  1.49995117e+01   1.47368164e+01   1.43789062e+01 ...,
    1.03535156e+01   1.01625977e+01   1.00571289e+01]]

[[  3.35302734e+00   3.52099609e+00   4.22607422e+00 ...,
    1.83056641e+00   1.85888672e+00   1.87011719e+00]
 [  4.57324219e+00   4.66601562e+00   5.28662109e+00 ...,
    1.79931641e+00   1.84179688e+00   1.89550781e+00]
 [  5.50292969e+00   5.90429688e+00   6.40087891e+00 ...,
    1.81054688e+00   1.87988281e+00   1.95556641e+00]
 ...,
 [  4.69726562e+00   4.63671875e+00   4.47802734e+00 ...,
    3.04248047e+00   3.00292969e+00   2.89794922e+00]
 [  4.69726562e+00   4.59033203e+00   4.46923828e+00 ...,
    3.16503906e+00   3.12548828e+00   2.99609375e+00]
 [  4.73730469e+00   4.64697266e+00   4.52636719e+00 ...,
    3.21337891e+00   3.14111328e+00   3.11181641e+00]]]
```

```
[[[  1.08653906e+03   1.16675000e+03   1.29650781e+03 ...,
     8.86500000e+02   9.00445312e+02   9.13171875e+02]
   [  1.25835156e+03   1.32050000e+03   1.41885938e+03 ...,
     8.99742188e+02   9.18125000e+02   9.93585938e+02]
   [  1.38512500e+03   1.47824219e+03   1.55324219e+03 ...,
     9.07671875e+02   9.35078125e+02   1.03723438e+03]
   ...,
   [  1.24385938e+03   1.23798438e+03   1.22862500e+03 ...,
     1.17260938e+03   1.16300000e+03   1.15532812e+03]
   [  1.24427344e+03   1.22956250e+03   1.22121094e+03 ...,
     1.18234375e+03   1.17617188e+03   1.16925781e+03]
   [  1.24715625e+03   1.23269531e+03   1.22026562e+03 ...,
     1.19719531e+03   1.18769531e+03   1.18296094e+03]]

  [[  1.06713281e+03   1.14761719e+03   1.27709375e+03 ...,
     8.68625000e+02   8.82453125e+02   8.95054688e+02]
   [  1.23951562e+03   1.30162500e+03   1.39900000e+03 ...,
     8.79890625e+02   9.00367188e+02   9.15000000e+02]
   [  1.36671094e+03   1.45803125e+03   1.53497656e+03 ...,
     8.89101562e+02   9.16078125e+02   9.38671875e+02]
   ...,
   [  1.22517969e+03   1.21956250e+03   1.21053125e+03 ...,
     1.15046094e+03   1.14350000e+03   1.13671094e+03]
   [  1.22558594e+03   1.21174219e+03   1.20142969e+03 ...,
     1.16225781e+03   1.15630469e+03   1.15171094e+03]
   [  1.22813281e+03   1.21738281e+03   1.20394531e+03 ...,
     1.17771875e+03   1.16957812e+03   1.16466406e+03]]

  [[  1.04629688e+03   1.12050000e+03   1.24889844e+03 ...,
     8.44304688e+02   8.57945312e+02   8.71007812e+02]
   [  1.21717188e+03   1.27964062e+03   1.37465625e+03 ...,
     8.55773438e+02   8.74835938e+02   8.88296875e+02]
   [  1.34216406e+03   1.43288281e+03   1.50972656e+03 ...,
     8.64382812e+02   8.90468750e+02   9.13601562e+02]
   ...,
   [  1.20134375e+03   1.19541406e+03   1.18455469e+03 ...,
     1.12676562e+03   1.11755469e+03   1.11141406e+03]
   [  1.20185156e+03   1.18594531e+03   1.17817188e+03 ...,
     1.13729688e+03   1.13100781e+03   1.12575000e+03]
   [  1.20676562e+03   1.19364062e+03   1.17849219e+03 ...,
     1.15292969e+03   1.14534375e+03   1.14100000e+03]]

  ...,
  [[  1.79506836e+01   1.89584961e+01   2.29765625e+01 ...,
     9.58886719e+00   9.85644531e+00   1.00439453e+01]
   [  2.47998047e+01   2.62729492e+01   2.95439453e+01 ...,
     9.89208984e+00   1.02319336e+01   1.02709961e+01]
   [  2.99829102e+01   3.29252930e+01   3.56660156e+01 ...,
```

```
      9.96777344e+00   1.05209961e+01   1.07265625e+01]
   ...,
   [  2.57949219e+01   2.59150391e+01   2.53964844e+01 ...,
      1.82070312e+01   1.77646484e+01   1.71923828e+01]
   [  2.58081055e+01   2.55898438e+01   2.51557617e+01 ...,
      1.87656250e+01   1.83276367e+01   1.77661133e+01]
   [  2.59941406e+01   2.55244141e+01   2.49541016e+01 ...,
      1.90659180e+01   1.86518555e+01   1.83647461e+01]]

  [[  9.91845703e+00   1.05009766e+01   1.27167969e+01 ...,
      5.30078125e+00   5.44238281e+00   5.56298828e+00]
   [  1.37163086e+01   1.45170898e+01   1.63378906e+01 ...,
      5.45947266e+00   5.64990234e+00   5.67773438e+00]
   [  1.65747070e+01   1.82236328e+01   1.97377930e+01 ...,
      5.47070312e+00   5.78613281e+00   5.95019531e+00]
   ...,
   [  1.42607422e+01   1.43691406e+01   1.40810547e+01 ...,
      1.00903320e+01   9.83935547e+00   9.53466797e+00]
   [  1.42734375e+01   1.41909180e+01   1.39462891e+01 ...,
      1.04052734e+01   1.01567383e+01   9.84423828e+00]
   [  1.43901367e+01   1.41142578e+01   1.38022461e+01 ...,
      1.05732422e+01   1.03251953e+01   1.01816406e+01]]

  [[  3.08740234e+00   3.28417969e+00   3.98388672e+00 ...,
      1.64843750e+00   1.69335938e+00   1.73339844e+00]
   [  4.27978516e+00   4.49658203e+00   5.09179688e+00 ...,
      1.65673828e+00   1.70361328e+00   1.77832031e+00]
   [  5.16210938e+00   5.67041016e+00   6.15722656e+00 ...,
      1.65917969e+00   1.75341797e+00   1.85009766e+00]
   ...,
   [  4.46191406e+00   4.50927734e+00   4.41113281e+00 ...,
      3.15332031e+00   3.08349609e+00   2.99707031e+00]
   [  4.46435547e+00   4.45312500e+00   4.37841797e+00 ...,
      3.25439453e+00   3.18798828e+00   3.10107422e+00]
   [  4.49414062e+00   4.41601562e+00   4.32373047e+00 ...,
      3.33642578e+00   3.26123047e+00   3.20703125e+00]]]


P array at time 0:
[[[  974.2734375   1009.46875     1183.6875     ...,   873.0703125
     883.3125       894.546875  ]
  [ 1224.4375      1245.109375    1352.4765625  ...,   893.5703125
     917.1640625    984.453125  ]
  [ 1393.328125    1445.6328125   1515.1640625  ...,   952.703125
    1056.8828125   1040.3828125 ]
  ...,
  [ 1267.1953125   1271.0390625   1261.5859375  ...,  1334.59375
    1338.53125     1344.1796875 ]
```

```
 [ 1246.3984375   1245.4375      1243.984375   ...,  1347.3515625
   1350.3828125   1355.4609375 ]
 [ 1240.0234375   1238.609375    1236.046875   ...,  1360.0234375
   1361.1328125   1366.03125   ]]

[[  954.5859375    991.15625     1163.609375   ...,   854.7109375
    864.90625      876.0390625 ]
 [ 1205.0703125   1223.546875    1331.3359375  ...,   874.6796875
    888.5546875    904.328125  ]
 [ 1374.0078125   1425.875       1495.640625   ...,   902.4453125
    921.6640625    938.984375  ]
 ...,
 [ 1249.8515625   1253.6171875   1243.765625   ...,  1314.0546875
   1318.109375    1323.90625   ]
 [ 1229.2734375   1228.03125     1226.1015625  ...,  1326.6796875   1330.
   1335.2734375 ]
 [ 1222.84375     1221.1796875   1218.2734375  ...,  1339.40625
   1340.703125    1345.78125   ]]

[[  929.390625     966.953125    1136.5546875  ...,   830.5625
    840.5859375    851.625     ]
 [ 1179.890625    1196.3046875   1304.84375    ...,   850.1875
    863.7890625    878.5625    ]
 [ 1347.9921875   1399.7265625   1470.1171875  ...,   876.7109375
    896.0234375    912.78125   ]
 ...,
 [ 1226.5546875   1229.96875     1220.1953125  ...,  1286.671875
   1290.9453125   1296.9765625 ]
 [ 1206.125       1204.609375    1202.4765625  ...,  1299.328125
   1302.90625     1308.359375  ]
 [ 1199.921875    1197.8515625   1194.796875   ...,  1311.9921875
   1313.5859375   1318.9609375 ]]


...,
[[   15.25292969    14.93896484    19.43310547 ...,     9.24658203
      9.51513672     9.71826172]
 [   23.99316406    23.65771484    26.93017578 ...,     9.61279297
      9.86816406    10.12646484]
 [   29.85449219    31.44189453    33.57324219 ...,    10.14453125
     10.49853516    10.75537109]
 ...,
 [   26.78466797    26.53027344    25.75488281 ...,    22.32958984
     22.21875       22.25341797]
 [   26.43994141    26.0390625     25.58203125 ...,    22.98632812
     22.85546875    22.88037109]
 [   26.59667969    26.19433594    25.72412109 ...,    23.55566406
     23.42529297    23.48779297]]
```

```
[[    8.44482422     8.27246094    10.76171875 ...,     5.12695312
      5.27636719     5.37158203]
 [   13.30175781    13.10302734    14.92333984 ...,     5.31494141
      5.46923828     5.61376953]
 [   16.54833984    17.41748047    18.60107422 ...,     5.61279297
      5.81933594     5.96337891]
 ...,
 [   14.83984375    14.703125      14.26757812 ...,    12.37695312
     12.30273438    12.32958984]
 [   14.64746094    14.42773438    14.17724609 ...,    12.73632812
     12.66650391    12.6796875 ]
 [   14.74121094    14.50683594    14.2421875  ...,    13.05615234
     12.97167969    13.01074219]]

[[    2.65966797     2.60253906     3.39355469 ...,     1.61523438
      1.65576172     1.69189453]
 [    4.18066406     4.13378906     4.70117188 ...,     1.67089844
      1.72070312     1.76171875]
 [    5.20361328     5.4921875      5.85595703 ...,     1.77587891
      1.83496094     1.87158203]
 ...,
 [    4.66845703     4.62988281     4.49121094 ...,     3.89599609
      3.87841797     3.87890625]
 [    4.62060547     4.54931641     4.46826172 ...,     4.00585938
      3.98095703     3.99169922]
 [    4.63720703     4.57568359     4.48974609 ...,     4.11523438
      4.08496094     4.09130859]]]
```

12.16 Pop Quiz

What is the first rule of data processing?

```
A) YOU DO NOT TALK ABOUT DATA PROCESSING
B) 60% OF THE TIME, IT WORKS EVERY TIME
C) ALWAYS LOOK AT YOUR DATA
```

13 3.0 wrf-python

wrf-python provides functionality similar to what is found in the NCL-WRF package:

- over 30 diagnostics calculations
- several interpolation routines (horizontal level, vertical cross section, horizontal "surface")
- plot helper utilities for cartopy, basemap, and PyNGL
- WRF-ARW only

The most commonly used functions:

- **getvar**: Extracts variables and diagnostic variables
- **interplevel**: Linearly interpolates a variable to a horizontal plane at a specified vertical level
- **vertcross**: Interpolates a 3D variable to a vertical cross section
- **vinterp**: Interpolates a variable to a new surface (e.g. theta-e)

13.1 The getvar function

The *getvar* function can be used to:

- Extract NetCDF variables from a file, similar to netcdf4-python or PyNIO.
- Compute diagnostic variables.
- Concatenate a variable (either NetCDF or diagnostic) across multiple files.

13.1.1 Simple *getvar* Example for HGT

```
from netCDF4 import Dataset
from wrf import getvar

file_path = "./wrfout_d01_2005-08-28_00:00:00"

wrf_file = Dataset(file_path)

hgt = getvar(wrf_file, "HGT", timeidx=0)
```

13.2 Your Turn!

13.2.1 Example 3.1: Using *getvar* to Extract a WRF NetCDF Variable

```
In [8]: from netCDF4 import Dataset
        from wrf import getvar

        file_path = single_wrf_file()

        wrf_file = Dataset(file_path)

        hgt = getvar(wrf_file, "HGT", timeidx=0)

        print(hgt)
```

```
<xarray.DataArray 'HGT' (south_north: 73, west_east: 90)>
array([[  9.154816e+02,   8.214922e+02,   1.082335e+03, ...,   0.000000e+00,
          1.553946e-02,   0.000000e+00],
       [  1.516390e+03,   1.452862e+03,   1.714327e+03, ...,   0.000000e+00,
          0.000000e+00,   0.000000e+00],
       [  1.986543e+03,   2.103951e+03,   2.308108e+03, ...,   1.500177e+01,
          2.438107e+01,   2.241381e+01],
```

```
       ...,
       [  9.018121e+02,   8.766862e+02,   8.207339e+02, ...,   1.801364e+01,
          1.330432e+01,   9.664661e+00],
       [  8.827278e+02,   8.420260e+02,   7.991212e+02, ...,   2.100833e+01,
          1.504651e+01,   1.044492e+01],
       [  9.039702e+02,   8.619205e+02,   8.150797e+02, ...,   2.742290e+01,
          1.620821e+01,   1.120480e+01]], dtype=float32)
Coordinates:
    XLONG    (south_north, west_east) float32 -101.008 -100.738 -100.468 ...
    XLAT     (south_north, west_east) float32 19.1075 19.1075 19.1075 ...
    XTIME    float32 0.0
    Time     datetime64[ns] 2005-08-28
Dimensions without coordinates: south_north, west_east
Attributes:
    FieldType:    104
    MemoryOrder:  XY
    description:  Terrain Height
    units:        m
    stagger:
    coordinates:  XLONG XLAT XTIME
    projection:   Mercator(stand_lon=-89.0, moad_cen_lat=27.9999923706, truel...
```

13.3 Computing a Diagnostic Variable with *getvar*

In this example, we're going to compute sea level pressure.

```
from netCDF4 import Dataset
from wrf import getvar

file_path = "./wrfout_d01_2005-08-28_00:00:00"

wrf_file = Dataset(file_path)

slp = getvar(wrf_file, "slp",
             timeidx=0, units="hPa")
```

Note the 'units' keyword argument. Some diagnostics support several choices for units. However, unit support is still relatively primitive.

13.4 Your Turn!

13.4.1 Example 3.2: Using *getvar* to compte Sea Level Pressure (SLP)

Also try changing the units for by specifiying the following values: 'hPa', 'Pa', 'atm', 'mmhg'

```
In [9]: from netCDF4 import Dataset
        from wrf import getvar
```

```
        file_path = single_wrf_file()

        wrf_file = Dataset(file_path)

        slp = getvar(wrf_file, "slp", timeidx=0, units="mmhg")

        print (slp)
```

```
<xarray.DataArray u'slp' (south_north: 73, west_east: 90)>
array([[ 756.113403,  756.487427,  757.554138, ...,  756.763306,  756.838867,
         756.922058],
       [ 757.443787,  757.628418,  758.183899, ...,  756.920105,  757.095947,
         757.599121],
       [ 758.39093 ,  758.692139,  759.045349, ...,  757.346802,  758.114746,
         757.987915],
       ...,
       [ 758.638733,  758.736877,  758.750305, ...,  760.176086,  760.207947,
         760.25354 ],
       [ 758.458557,  758.536438,  758.609192, ...,  760.270569,  760.296082,
         760.337463],
       [ 758.358154,  758.436768,  758.51001 , ...,  760.364807,  760.375671,
         760.415955]], dtype=float32)
Coordinates:
    XLONG    (south_north, west_east) float32 -101.008 -100.738 -100.468 ...
    XLAT     (south_north, west_east) float32 19.1075 19.1075 19.1075 ...
    XTIME    float32 0.0
    Time     datetime64[ns] 2005-08-28
Dimensions without coordinates: south_north, west_east
Attributes:
    FieldType:    104
    MemoryOrder:  XY
    description:  sea level pressure
    units:        mmhg
    stagger:
    coordinates:  XLONG XLAT XTIME
    projection:   Mercator(stand_lon=-89.0, moad_cen_lat=27.9999923706, truel...
```

13.5 Combining Across Mulitple Files

wrf-python has two methods for combining a variable across multiple files

- **cat** - combines the the variable along the Time dimension (Note: you must order the files yourself)
- **join** - creates a new left-most dimension for each file

To extract all times in to a single array, set *timeidx* to wrf.ALL_TIMES (an alias for None). In this example, we're using the 'cat' method, which is the most common.

```
from netCDF4 import Dataset
from wrf import getvar, ALL_TIMES

file_paths = [
     "./wrfout_d01_2005-08-28_00:00:00",
     "./wrfout_d01_2005-08-28_12:00:00"
     ]

wrf_files = [Dataset(file_paths[0]),
             Dataset(file_paths[1])]

slp = getvar(wrf_files, "slp",
             timeidx=ALL_TIMES,
             method="cat")
```

13.6 Your Turn!

13.6.1 Example 3.3: Combining Files Using the 'cat' Method

```
In [10]: from netCDF4 import Dataset
         from wrf import getvar, ALL_TIMES

         file_paths = multiple_wrf_files()

         wrf_files = [Dataset(f) for f in file_paths]

         slp = getvar(wrf_files, "slp", timeidx=ALL_TIMES, method="cat")

         print (slp)
```

```
<xarray.DataArray u'slp' (Time: 9, south_north: 73, west_east: 90)>
array([[[ 1008.068115,  1008.566772, ...,  1009.035339,  1009.14624 ],
        [ 1009.841858,  1010.088013, ...,  1009.378052,  1010.04895 ],
        ...,
        [ 1011.194702,  1011.298584, ...,  1013.644531,  1013.699768],
        [ 1011.060852,  1011.16571 , ...,  1013.750671,  1013.804382]],

       [[ 1009.050232,  1009.908264, ...,  1009.286438,  1009.375   ],
        [ 1010.572144,  1010.996887, ...,  1009.511475,  1010.215454],
        ...,
        [ 1011.312988,  1011.194214, ...,  1012.642212,  1012.711914],
        [ 1011.253418,  1011.26416 , ...,  1012.804382,  1012.82373 ]],

       ...,
       [[ 1007.129578,  1007.888184, ...,  1009.449341,  1009.516541],
        [ 1008.402466,  1008.880005, ...,  1009.630981,  1010.418579],
        ...,
        [ 1009.997314,  1010.131104, ...,  1011.760376,  1011.681152],
        [ 1010.18573 ,  1010.160522, ...,  1011.766846,  1011.776001]],
```

```
       [[ 1005.577087,  1006.532715, ...,  1008.739441,  1008.798706],
        [ 1006.842529,  1007.557495, ...,  1008.946045,  1009.73761 ],
        ...,
        [ 1008.742676,  1009.442017, ...,  1011.559631,  1011.502563],
        [ 1009.131042,  1009.176819, ...,  1011.558411,  1011.555481]]], dtype=float32)
Coordinates:
    XLONG     (south_north, west_east) float32 -101.008 -100.738 -100.468 ...
    XLAT      (south_north, west_east) float32 19.1075 19.1075 19.1075 ...
    XTIME     (Time) float64 0.0 180.0 360.0 540.0 720.0 900.0 1.08e+03 ...
  * Time      (Time) datetime64[ns] 2005-08-28 2005-08-28T03:00:00 ...
    datetime  (Time) datetime64[ns] 2005-08-28 2005-08-28T03:00:00 ...
Dimensions without coordinates: south_north, west_east
Attributes:
    FieldType:    104
    MemoryOrder:  XY
    description:  sea level pressure
    units:        hPa
    stagger:
    coordinates:  XLONG XLAT XTIME
    projection:   Mercator(stand_lon=-89.0, moad_cen_lat=27.9999923706, truel...
```

13.6.2 Example 3.4: Combining Files Using the 'join' Method

```
In [11]: from netCDF4 import Dataset
         from wrf import getvar, ALL_TIMES

         file_paths = multiple_wrf_files()

         wrf_files = [Dataset(f) for f in file_paths]

         slp = getvar(wrf_files, "slp", timeidx=ALL_TIMES, method="join")

         print (slp)
```

```
<xarray.DataArray u'slp' (file: 3, Time: 4, south_north: 73, west_east: 90)>
array([[[[ 1008.068115, ...,  1009.14624 ],
         ...,
         [ 1011.060852, ...,  1013.804382]],

        ...,
        [[ 1009.626709, ...,  1009.320251],
         ...,
         [ 1011.33197 , ...,  1011.954163]]],


       ...,
```

```
       [[[ 1005.577087, ...,  1008.798706],
         ...,
         [ 1009.131042, ...,  1011.555481]],

        ...,
        [[         nan, ...,          nan],
         ...,
         [         nan, ...,          nan]]]], dtype=float32)
Coordinates:
    XLONG     (south_north, west_east) float32 -101.008 -100.738 -100.468 ...
    XLAT      (south_north, west_east) float32 19.1075 19.1075 19.1075 ...
    XTIME     (file, Time) float32 0.0 180.0 360.0 540.0 720.0 900.0 1080.0 ...
  * file      (file) int64 0 1 2
    datetime  (file, Time) datetime64[ns] 2005-08-28 2005-08-28T03:00:00 ...
Dimensions without coordinates: Time, south_north, west_east
Attributes:
    FieldType:      104
    MemoryOrder:    XY
    description:    sea level pressure
    units:          hPa
    stagger:
    coordinates:    XLONG XLAT XTIME
    projection:     Mercator(stand_lon=-89.0, moad_cen_lat=27.9999923706, tru...
    _FillValue:     9.96921e+36
    missing_value:  9.96921e+36
```

13.7 Interpolation Routines

- **interplevel** - linear interpolation to a horizontal plane at a specified height or pressure level
- **vertcross** - vertical cross section interpolation to a vertical plane through two specified points (or a pivot point and angle)
- **vinterp** - interpolates to a "surface", which could be pressure levels or temperature levels like theta-e

13.8 The *interplevel* function

- The easiest way to get a field at a specified height or pressure vertical level (500 mb, 5000 m, etc).
- Uses linear interpolation, which is fastest and generally "good enough" for plotting.
- You should use *vinterp* if you want more accuracy, since the interpolation is done with the decaying exponential pressure profile.

13.8.1 *interplevel* Example

Let's get the 500 hPa geopotential height in decameters

```
from netCDF4 import Dataset
```

```
from wrf import getvar, interplevel

file_path = "./wrfout_d01_2005-08-28_00:00:00"

wrf_file = Dataset(file_path)

pres = getvar(wrf_file, "pressure", timeidx=0)

ht = getvar(wrf_file, "z", timeidx=0,
            units="dm")

ht_500 = interplevel(ht, pres, 500.0)
```

13.9 Your Turn!

13.9.1 Example 3.5: Interpolate to 500 hPa Using *interplevel*

```
In [12]: from netCDF4 import Dataset
         from wrf import getvar, interplevel

         file_path = single_wrf_file()

         wrf_file = Dataset(file_path)

         pres = getvar(wrf_file, "pressure", timeidx=0)
         ht = getvar(wrf_file, "z", timeidx=0, units="dm")

         ht_500 = interplevel(ht, pres, 500.0)

         print (ht_500)
```

```
<xarray.DataArray u'height_500.0_hPa' (south_north: 73, west_east: 90)>
array([[ 591.048035,  592.139771,  592.430969, ...,  589.43219 ,  589.29364 ,
         589.18512 ],
       [ 589.145081,  590.165588,  590.649658, ...,  589.607666,  589.511963,
         589.462708],
       [ 589.765625,  590.008789,  589.82959 , ...,  589.761475,  589.64679 ,
         589.592224],
       ...,
       [ 590.164978,  590.027466,  589.911743, ...,  588.630371,  588.646484,
         588.647583],
       [ 590.170288,  590.069519,  589.95697 , ...,  588.578125,  588.59137 ,
         588.589783],
       [ 590.143494,  590.054932,  589.952698, ...,  588.493896,  588.497131,
         588.486328]], dtype=float32)
Coordinates:
    XLONG    (south_north, west_east) float32 -101.008 -100.738 -100.468 ...
    XLAT     (south_north, west_east) float32 19.1075 19.1075 19.1075 ...
```

```
    XTIME    float32 0.0
    Time     datetime64[ns] 2005-08-28
Dimensions without coordinates: south_north, west_east
Attributes:
    FieldType:      104
    units:          dm
    stagger:
    coordinates:    XLONG XLAT XTIME
    projection:     Mercator(stand_lon=-89.0, moad_cen_lat=27.9999923706, tru...
    level:          500.0 hPa
    missing_value:  9.96920996839e+36
    _FillValue:     9.96920996839e+36
```

13.10 The *vertcross* function

The idea is to draw a horizontal line at the surface, and the cross section is defined as a vertical plane extending up from this line.

- The new 'x-axis' in the cross section is the points along the line you made. The line can be defined by:

1. defining a start point and an end point by using (x,y) grid coordinates or (latitude, longitude) coordinates.
2. defining a pivot point and an angle, which is useful for cross sections that will span most of the domain.

- The new 'y-axis' will be a set of vertical levels at default intervals, or you can choose them.

13.10.1 Introducing the *CoordPair* class

A *CoordPair* is simply used to store (x,y) coordinates, or (lat,lon) coordinates. It is also possible to have (x, y, lat, lon), but that's rarely used.

The *CoordPair* will be used to define your cross section line.

```
from wrf import CoordPair

# Creating an x,y pair
x_y_pair = CoordPair(x=10, y=20)

# Creating a lat,lon pair
lat_lon_pair = CoordPair(lat=30.0, lon=-120.0)
```

13.10.2 *vertcross* Example

In this example, we're going to define the cross section using a start point and and end point.

We're going to let the algorithm pick the levels, which are at ~1% increments by default.

```
from netCDF4 import Dataset
from wrf import getvar, vertcross, CoordPair

file_path = "./wrfout_d01_2005-08-28_00:00:00"
wrf_file = Dataset(file_path)

# Making a diagonal cross section line from
# bottom left to top right.
bottom_left = CoordPair(x=0, y=0)

top_right = CoordPair(x=-1, y=-1)

# Let's get wind speed in kts
wspd_wdir = getvar(wrf_file, "wspd_wdir",
                   timeidx=0, units="kt")

wspd = wspd_wdir[0,:]

# Get the height levels
ht = getvar(wrf_file, "z", timeidx=0)

wspd_cross = vertcross(wspd, ht,
               start_point=bottom_left,
               end_point=top_right)
```

13.11 Your Turn!

13.11.1 Example 3.6: Interpolate to a Vertical Cross Section with *vertcross*

```
In [13]: from netCDF4 import Dataset
         from wrf import getvar, vertcross, CoordPair

         file_path = single_wrf_file()
         wrf_file = Dataset(file_path)

         # Making a diagonal cross section line from
         # bottom left to top right.
         bottom_left = CoordPair(x=0, y=0)

         top_right = CoordPair(x=-1, y=-1)

         # Let's get wind speed in kts
         wspd_wdir = getvar(wrf_file, "wspd_wdir",
                            timeidx=0, units="kt")

         wspd = wspd_wdir[0,:]
```

```
        # Get the height levels
        ht = getvar(wrf_file, "z", timeidx=0)

        wspd_cross = vertcross(wspd, ht,
                               start_point=bottom_left,
                               end_point=top_right)

        print (wspd_cross)
```

```
<xarray.DataArray u'wspd_wdir_cross' (vertical: 100, cross_line_idx: 114)>
array([[       nan,        nan,        nan, ...,        nan,        nan,
               nan],
       [       nan,        nan,        nan, ...,   6.497844,   6.658993,
          7.310721],
       [       nan,        nan,        nan, ...,   8.698407,   9.173474,
         10.298722],
       ...,
       [ 30.791243,  30.527824,  30.21583 , ...,   9.199562,   9.146236,
          9.104509],
       [ 32.16003 ,  31.79801 ,  31.464676, ...,   9.679264,   9.643409,
          9.628514],
       [       nan,  33.068195,  32.713524, ...,  10.158965,  10.140581,
         10.152519]], dtype=float32)
Coordinates:
    wspd_wdir  <U4 u'wspd'
    XTIME      float32 0.0
    Time       datetime64[ns] 2005-08-28
    xy_loc     (cross_line_idx) object CoordPair(x=0.0, y=0.0) ...
  * vertical   (vertical) float32 0.0 204.947 409.895 614.842 819.789 ...
Dimensions without coordinates: cross_line_idx
Attributes:
    FieldType:      104
    description:    wspd,wdir in projection space
    units:          kt
    stagger:
    coordinates:    XLONG XLAT XTIME
    projection:     Mercator(stand_lon=-89.0, moad_cen_lat=27.9999923706, tru...
    orientation:    (0.0, 0.0) to (88.2192982456, 71.3684210526)
    missing_value:  9.96920996839e+36
    _FillValue:     9.96920996839e+36
```

13.12 The *vinterp* function

- Used for interpolating a field to a type of surface:
- pressure
- geopotential height

- theta
- theta-e
- User must specify the interpolation level(s) on the new surface
- A smarter, albeit slower and more complicated, version of *interplevel*

13.12.1 *vinterp* Example

In this example, we're going to interpolate pressure to theta-e levels.

```
from netCDF4 import Dataset
from wrf import getvar, vinterp

file_path = "./wrfout_d01_2005-08-28_00:00:00"
wrf_file = Dataset(file_path)

pres = getvar(wrf_file, "pressure", timeidx=0)

# Interpolate pressure to theta-e levels ă ă ă ă ă ă ă ă
interp_levels = [280, 285, 290, 292, 294,
                 296, 298, 300, 305, 310]

pres_eth = vinterp(wrf_file,
               field=pres,
               vert_coord="theta-e",
               interp_levels=interp_levels,
               extrapolate=False,
               field_type="pressure",
               log_p=False)
```

13.13 Your Turn!

13.13.1 Example 3.7: Interpolate to Theta-e Levels with *vinterp*

```
In [14]: from netCDF4 import Dataset
         from wrf import getvar, vinterp

         file_path = single_wrf_file()
         wrf_file = Dataset(file_path)

         pres = getvar(wrf_file, "pressure", timeidx=0)

         # Interpolate pressure to theta-e levels ă ă ă ă ă ă ă ă
         interp_levels = [280., 285., 290., 292., 294.,
                          296., 298., 300., 305., 310.]

         pres_eth = vinterp(wrf_file,
                        field=pres,
                        vert_coord="theta-e",
                        interp_levels=interp_levels,
```

```
                        extrapolate=True,
                        field_type="pressure",
                        log_p=False,
                        timeidx=0)

        print (pres_eth)
```

```
<xarray.DataArray u'pressure' (interp_level: 10, south_north: 73, west_east: 90)>
array([[[  903.574585,   914.063232, ...,  1005.506226,  1005.620422],
        [  843.457947,   850.108704, ...,  1005.846619,  1006.519531],
        ...,
        [  909.817505,   914.200623, ...,  1008.412781,  1009.003479],
        [  907.468628,   911.982788, ...,  1008.384216,  1009.019958]],

       [[  903.574585,   914.063232, ...,  1005.506226,  1005.620422],
        [  843.457947,   850.108704, ...,  1005.846619,  1006.519531],
        ...,
        [  909.817505,   914.200623, ...,  1008.412781,  1009.003479],
        [  907.468628,   911.982788, ...,  1008.384216,  1009.019958]],

       ...,
       [[  903.574585,   914.063232, ...,  1005.506226,  1005.620422],
        [  843.457947,   850.108704, ...,  1005.846619,  1006.519531],
        ...,
        [  909.817505,   914.200623, ...,  1008.412781,  1009.003479],
        [  907.468628,   911.982788, ...,  1008.384216,  1009.019958]],

       [[  903.574585,   914.063232, ...,  1005.506226,  1005.620422],
        [  843.457947,   850.108704, ...,  1005.846619,  1006.519531],
        ...,
        [  909.817505,   914.200623, ...,  1008.412781,  1009.003479],
        [  907.468628,   911.982788, ...,  1008.384216,  1009.019958]]], dtype=float32)
Coordinates:
    XLONG         (south_north, west_east) float32 -101.008 -100.738 ...
    XLAT          (south_north, west_east) float32 19.1075 19.1075 19.1075 ...
    XTIME         float32 0.0
    Time          datetime64[ns] 2005-08-28
  * interp_level  (interp_level) float64 280.0 285.0 290.0 292.0 294.0 296.0 ...
Dimensions without coordinates: south_north, west_east
Attributes:
    FieldType:         104
    MemoryOrder:       XYZ
    description:       pressure
    units:             hPa
    stagger:
    coordinates:       XLONG XLAT XTIME
    projection:        Mercator(stand_lon=-89.0, moad_cen_lat=27.9999923706, ...
    vert_interp_type:  theta-e
```

14 Other Useful Functions

14.1 *to_np*

Converts an xarray.DataArray to a numpy array.

This is often necessary when passing wrf-python variables to the plotting functions or to a compiled extension module.

This routines does the following:

- If no missing values, then it simply calls the *xarray.DataArray.values* property.
- If missing values are found, then it replaces the NaN that xarray uses for missing values with the fill value found in the attributes.

14.1.1 *to_np* Example

```
from netCDF4 import Dataset
from wrf import getvar, to_np

file_path = "./wrfout_d01_2005-08-28_00:00:00"
wrf_file = Dataset(file_path)

pres_xarray = getvar(wrf_file, "pressure", timeidx=0)

pres_numpy = to_np(pres_xarray)
```

14.2 xy_to_ll and ll_to_xy

These routines convert to/from grid (x,y) coordinates to/from (lat,lon) coordinates.

Works with a single point or sequences of points.

14.2.1 xy_to_ll and ll_to_xy Example

```
from netCDF4 import Dataset
from wrf import getvar, xy_to_ll, ll_to_xy

file_path = "./wrfout_d01_2005-08-28_00:00:00"
wrf_file = Dataset(file_path)

lat_lon = xy_to_ll(wrf_file,

                   [20,30],
                   [50,75])

x_y = ll_to_xy(wrf_file,
```

```
                lat_lon[0,:],
                lat_lon[1,:])
```

14.3 Your Turn!

14.3.1 Example 3.8: *xy_to_ll* and *ll_to_xy*

```
In [15]: from netCDF4 import Dataset
         from wrf import getvar, xy_to_ll, ll_to_xy

         file_path = single_wrf_file()
         wrf_file = Dataset(file_path)

         lat_lon = xy_to_ll(wrf_file, [20, 30], [50,75])

         print ("lat,lon values")
         print (lat_lon)
         print ("\n")

         x_y = ll_to_xy(wrf_file, lat_lon[0,:], lat_lon[1,:])

         print ("x,y values")
         print (x_y)
```

```
lat,lon values
<xarray.DataArray u'latlon' (lat_lon: 2, idx: 2)>
array([[ 31.282637,  36.86616 ],
       [-95.611051, -92.912663]])
Coordinates:
  * lat_lon   (lat_lon) <U3 u'lat' u'lon'
    xy_coord  (idx) object CoordPair(x=20, y=50) CoordPair(x=30, y=75)
Dimensions without coordinates: idx


x,y values
<xarray.DataArray u'xy' (x_y: 2, idx: 2)>
array([[20, 30],
       [50, 75]])
Coordinates:
    latlon_coord  (idx) object CoordPair(lat=31.2826368543, lon=-95.6110510386) ...
  * x_y           (x_y) <U1 u'x' u'y'
Dimensions without coordinates: idx
```

15 4.0 Plotting

15.1 Plotting

- Plotting wrf-python variables in Python can be done using either matplotlib or PyNGL.

- For this tutorial, we're going to focus on matplotlib (using cartopy for the mapping).
- We're going to use matplotlib 1.5.3, since cartopy is still having some issues with matplotlib 2.x.

15.2 Extremely Brief Overview of Matplotlib

- Under the hood, matplotlib uses an object oriented API.
- In simplest terms, a matplotlib plot consists of Figure object that contains an Axes object (or multiple Axes objects).
- The Axes object is where the action is.
- The Axes object contains the methods for contouring, making histograms, adding colorbar, etc.

15.2.1 The pyplot API

- Most matplotlib users do no need to know much about the underlying object oriented API.
- Matplotlib includes a series of standalone functions that wrap around the object oriented API.
- These standalone functions are in the matplotlib.pyplot package and it was designed to look similar to the Matlab API.

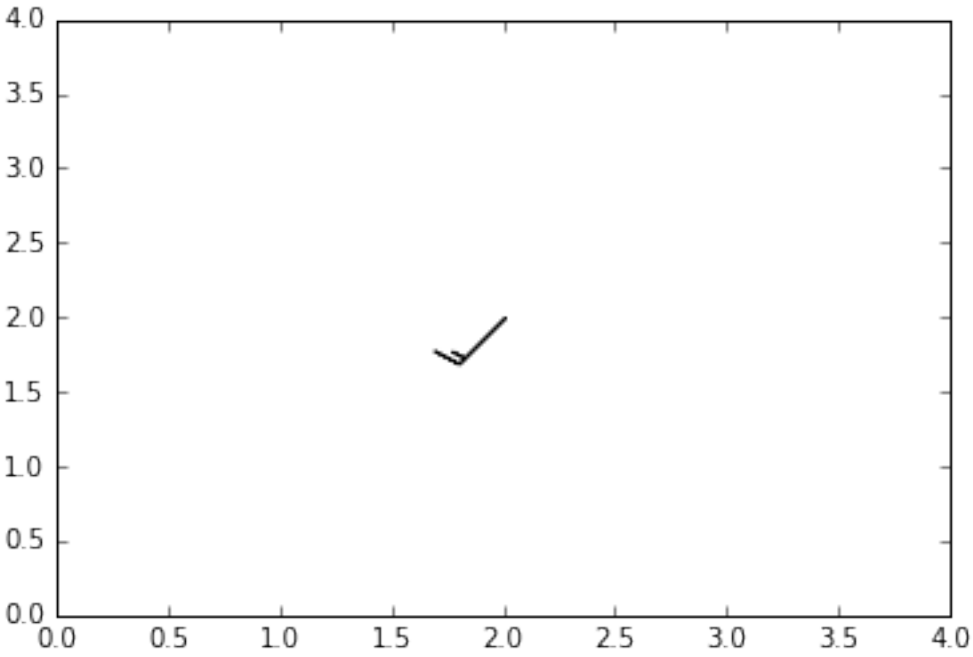
15.3 Your Turn!

15.3.1 Example 4.1: Single Wind Barb Example with pyplot

In this example, we are going to plot a single wind barb in the center of the domain using the pyplot API.

```
In [16]: from matplotlib import pyplot
         import numpy as np

         # Make a 5x5 grid of missing u,v values
         u = np.ma.masked_equal(np.zeros((5,5)), 0)
         v = np.ma.masked_equal(np.zeros((5,5)), 0)

         # Add u,v winds to center of domain
         u[2,2] = 10.0
         v[2,2] = 10.0

         # Draw a single wind barb in the middle using pyplot API
         # Note:  the axes objects are "hidden" in these functions
         fig = pyplot.figure()
         pyplot.barbs(u, v)

         # Set the x and y ranges so the barb is in the middle
```

```
pyplot.xlim(0, 4)
pyplot.ylim(0, 4)

pyplot.show()
```



15.4 Mixing the APIs

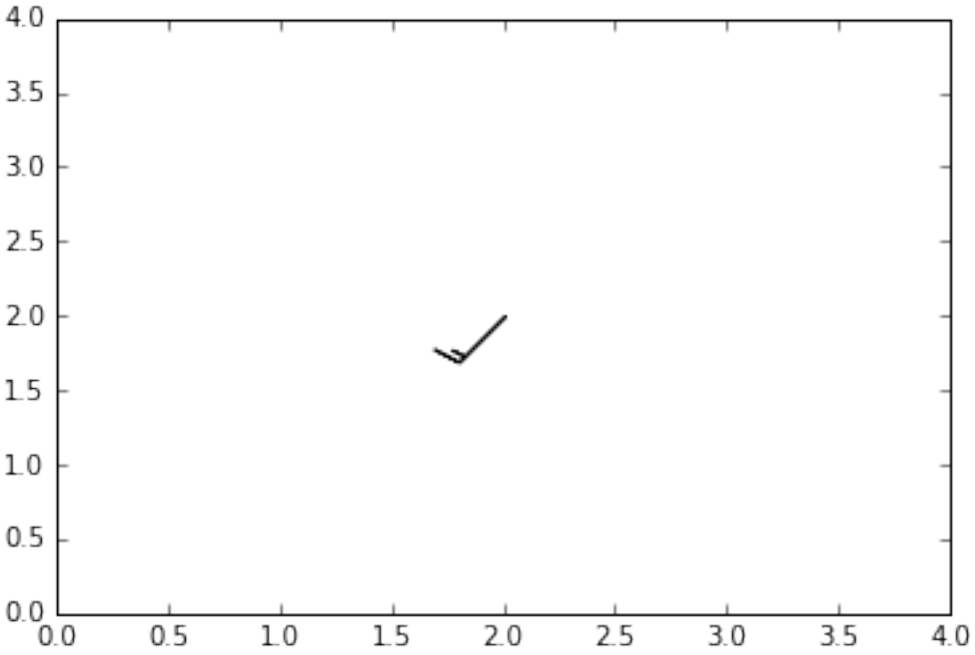
- Often you will find yourself mixing the object oriented API with the pyplot API.
- This is required when making subplots, but that is beyond the scope of this tutorial.
- The next example shows how to make the single wind barb using the axes object directly.

15.5 Your Turn!

15.5.1 Example 4.2: Single Wind Barb Using the Axes Object

```
In [17]: from matplotlib import pyplot
         import numpy as np

         # Make a 5x5 grid of missing u,v values
         u = np.ma.masked_equal(np.zeros((5,5)), 0)
         v = np.ma.masked_equal(np.zeros((5,5)), 0)

         # Add u,v winds to center of domain
         u[2,2] = 10.0
```

```
v[2,2] = 10.0

# We'll use pyplot to create the figure and
# get the axes
fig = pyplot.figure()
ax = pyplot.axes() # <- Remember this line

# Now use the axes directly to create the barbs
ax.barbs(u, v)

# Set the x and y ranges using the axes directly
ax.set_xlim(0, 4)
ax.set_ylim(0, 4)

pyplot.show()
```



15.6 wrf-python Plotting Helper Functions

wrf-python has several functions to help with plotting when using cartopy, basemap, or PyNGL.

- **get_cartopy, get_basemap, get_pyngl**: Returns the mapping object used by the plotting system
- **latlon_coords**: Returns the latitude and longitude coordinate variables
- **get_bounds**: Returns the geographic boundaries for the variable

15.7 Plotting with cartopy

Cartopy uses the same API as matplotlib by returning a matplotlib.axes.Axes subclass (cartopy.mpl.geoaxes.GeoAxes) when a *projection* keyword argument is passed to the matplotlib.pyplot.axes function.

15.7.1 Getting the GeoAxes object

```
import matplotlib.pyplot
import cartopy.crs

# Set up a standard map for latlon data.
latlon = cartopy.crs.PlateCarree()

geo_axes = pyplot.axes(projection=latlon)
```

15.7.2 Getting the cartopy projection object and lat,lon coordinates using wrf-python

- When xarray is installed and enabled, wrf-python carries the projection information around in the metadata of a variable.
- You can use the *get_cartopy* function to extract the cartopy projection object from a variable.
- You can use the *latlon_coords* function to get the latitude and longitude points.

```
from netCDF4 import Dataset
from wrf import (getvar, get_cartopy,
                 latlon_coords)

file_path = "./wrfout_d01_2005-08-28_00:00:00"

wrf_file = Dataset(file_path)

terrain = getvar(wrf_file, "ter", timeidx=0)

cart_proj = get_cartopy(terrain)

lats, lons = latlon_coords(terrain)
```

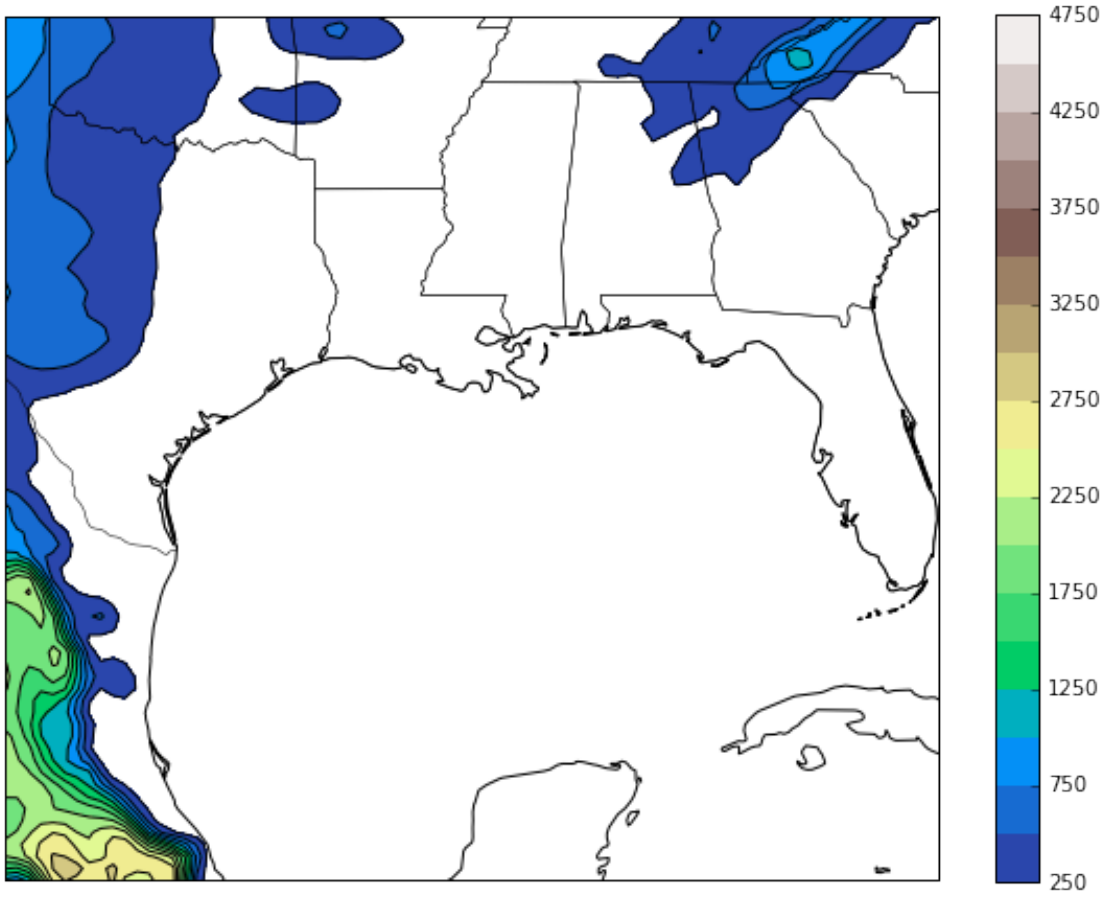
15.8 Your Turn!

15.8.1 Example 4.3: Making a Plot of Terrain

Let's make a plot of terrain. It's the easiest way to check if your map is correct.

```
In [18]: import numpy
         from matplotlib import pyplot
         from matplotlib.cm import get_cmap
         from cartopy import crs
         from cartopy.feature import NaturalEarthFeature
```

```
from netCDF4 import Dataset
from wrf import getvar, to_np, get_cartopy, latlon_coords

file_path = single_wrf_file()
wrf_file = Dataset(file_path)

# Get the terrain height
terrain = getvar(wrf_file, "ter", timeidx=0)

# Get the cartopy object and the lat,lon coords
cart_proj = get_cartopy(terrain)
lats, lons = latlon_coords(terrain)

# Create a figure and get the GetAxes object
fig = pyplot.figure(figsize=(10, 7.5))
geo_axes = pyplot.axes(projection=cart_proj)

# Download and add the states and coastlines
# See the cartopy documentation for more on this.
states = NaturalEarthFeature(category='cultural',
                             scale='50m',
                             facecolor='none',
                             name='admin_1_states_provinces_shp')
geo_axes.add_feature(states, linewidth=.5)
geo_axes.coastlines('50m', linewidth=0.8)

# Set the contour levels
levels = numpy.arange(250., 5000., 250.)

# Make the contour lines and fill them.
pyplot.contour(to_np(lons), to_np(lats),
               to_np(terrain), levels=levels,
               colors="black",
               transform=crs.PlateCarree())
pyplot.contourf(to_np(lons), to_np(lats),
                to_np(terrain), levels=levels,
                transform=crs.PlateCarree(),
                cmap=get_cmap("terrain"))

# Add a color bar. The shrink often needs to be set
# by trial and error.
pyplot.colorbar(ax=geo_axes, shrink=.99)

pyplot.show()
```



15.9 Cropping

Sometimes WRF domains are much larger than what you care about.

Plots can be cropped in two ways using wrf-python:

1. Crop the data before plotting.

- Less data to process = faster!
- There's a slight risk of issues at borders, but matplotlib seems pretty smart about this.

2. Crop the domain with matplotlib using x,y axis limits.

- Runs slower since all of the domain is contoured.
- Should always be correct.

15.9.1 Method 1: Cropping then Plotting

If you are cropping the data, then cartopy should just work without worrying about setting the axis limits, as long as xarray is installed and enabled.

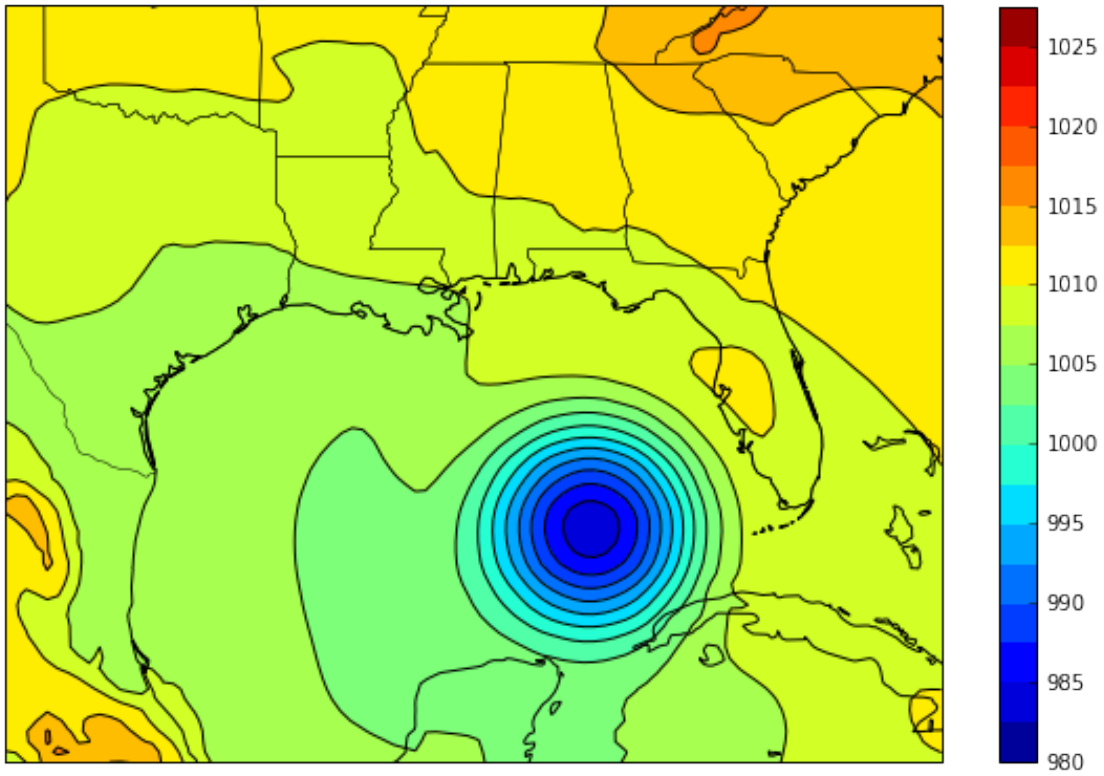
Let's start by taking a quick look at the full plot of sea level pressure.

15.10 Your Turn!

15.10.1 Example 4.4: Full Plot of Sea Level Pressure

In [19]:
```
import numpy
from matplotlib import pyplot
from matplotlib.cm import get_cmap
from cartopy import crs
from cartopy.feature import NaturalEarthFeature
from netCDF4 import Dataset
from wrf import getvar, to_np, get_cartopy, latlon_coords

file_path = single_wrf_file()
wrf_file = Dataset(file_path)

# Get the terrain height
slp = getvar(wrf_file, "slp", timeidx=0)

# Get the cartopy object and the lat,lon coords
cart_proj = get_cartopy(slp)
lats, lons = latlon_coords(slp)

# Create a figure and get the GetAxes object
fig = pyplot.figure(figsize=(10, 7.5))
geo_axes = pyplot.axes(projection=cart_proj)

# Download and add the states and coastlines
# See the cartopy documentation for more on this.
states = NaturalEarthFeature(category='cultural',
                             scale='50m',
                             facecolor='none',
                             name='admin_1_states_provinces_shp')
geo_axes.add_feature(states, linewidth=.5)
geo_axes.coastlines('50m', linewidth=0.8)

# Set the contour levels so that all plots match
levels = numpy.arange(980.,1030.,2.5)

# Make the contour lines and fill them.
pyplot.contour(to_np(lons), to_np(lats),
               to_np(slp), levels=levels, colors="black",
               transform=crs.PlateCarree())
pyplot.contourf(to_np(lons), to_np(lats),
                to_np(slp), levels=levels,
                transform=crs.PlateCarree(),
                cmap=get_cmap("jet"))

# Add a color bar. The shrink often needs to be set
```

```
# by trial and error.
pyplot.colorbar(ax=geo_axes, shrink=.86)

pyplot.show()
```



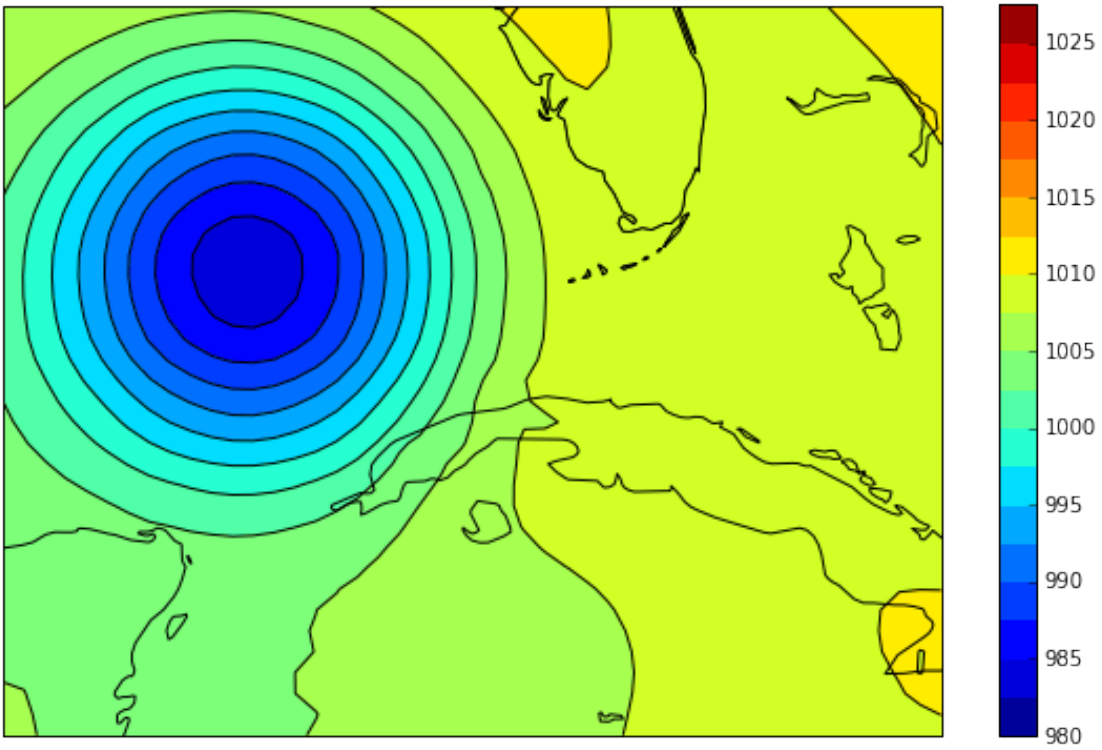
15.11 Your Turn!

15.11.1 Example 4.5: Cropping by Slicing the Data

Let's crop the data to the lower right quadrant.

```
In [20]: import numpy
         from matplotlib import pyplot
         from matplotlib.cm import get_cmap
         from cartopy import crs
         from cartopy.feature import NaturalEarthFeature
         from netCDF4 import Dataset
         from wrf import getvar, to_np, get_cartopy, latlon_coords

         file_path = single_wrf_file()
         wrf_file = Dataset(file_path)

         # Get the terrain height
```

```
slp = getvar(wrf_file, "slp", timeidx=0)

# Determine the center of the domain in grid coordinates
slp_shape = slp.shape
center_y = int(slp_shape[-2]/2.) - 1
center_x = int(slp_shape[-1]/2.) - 1

# Slice from bottom to middle for y
# Slice from middle to right for x
slp_quad = slp[..., 0:center_y+1, center_x:]

# Get the cartopy object and the lat,lon coords
cart_proj = get_cartopy(slp_quad)
lats, lons = latlon_coords(slp_quad)

# Create a figure and get the GetAxes object
fig = pyplot.figure(figsize=(10, 7.5))
geo_axes = pyplot.axes(projection=cart_proj)

# Download and add the states and coastlines
# See the cartopy documentation for more on this.
states = NaturalEarthFeature(category='cultural',
                             scale='50m',
                             facecolor='none',
                             name='admin_1_states_provinces_shp')
geo_axes.add_feature(states, linewidth=.5)
geo_axes.coastlines('50m', linewidth=0.8)

# Set the contour levels so that all plots match
levels = numpy.arange(980.,1030.,2.5)

# Make the contour lines and fill them.
pyplot.contour(to_np(lons), to_np(lats),
               to_np(slp_quad), levels=levels, colors="black",
               transform=crs.PlateCarree())
pyplot.contourf(to_np(lons), to_np(lats),
                to_np(slp_quad), levels=levels,
                transform=crs.PlateCarree(),
                cmap=get_cmap("jet"))

# Add a color bar. The shrink often needs to be set
# by trial and error.
pyplot.colorbar(ax=geo_axes, shrink=.83)

pyplot.show()
```



15.11.2 Method 2: Cropping by Setting x and y Extents

This time, let's crop the domain by using the x and y extents in matplotlib.

Also, we're going to crop the domain using lat,lon geographic boundaries.

15.11.3 Introducing the GeoBounds class

To create geographic boundaries, you supply a GeoBounds object constructed with set of bottom_left and top_right CoordPair objects.

The CoordPair objects need to use the *lat* and *lon* arguments.

```
from wrf import CoordPair, GeoBounds

bottom_left = CoordPair(lat=29.5, lon=-110)
top_right = CoordPair(lat=30.0, lon=-109.3)

geo_bounds = GeoBounds(bottom_left, top_right)
```

15.11.4 Setting the Cartopy Extents

After setting up the GeoBounds objects, you can use *cartopy_xlim* and *cartopy_ylim* functions to set the extents.

Note: Cartopy also has an API for doing this, but it doesn't work correctly for some projections like the RotatedPole.

```
from wrf import (CoordPair, GeoBounds,
                 getvar, cartopy_xlim,
                 cartopy_ylim)

bottom_left = CoordPair(lat=29.5, lon=-110)
top_right = CoordPair(lat=30.0, lon=-109.3)

geo_bounds = GeoBounds(bottom_left, top_right)
.
. (Set up variable, figure, geo_axes, etc)
.

xlim = cartopy_xlim(variable,
                    geobounds=geo_bounds))

geo_axes.set_xlim(xlim)

ylim = cartopy_ylim(variable,
                    geobounds=geo_bounds))

geo_axes.set_ylim(ylim)
```

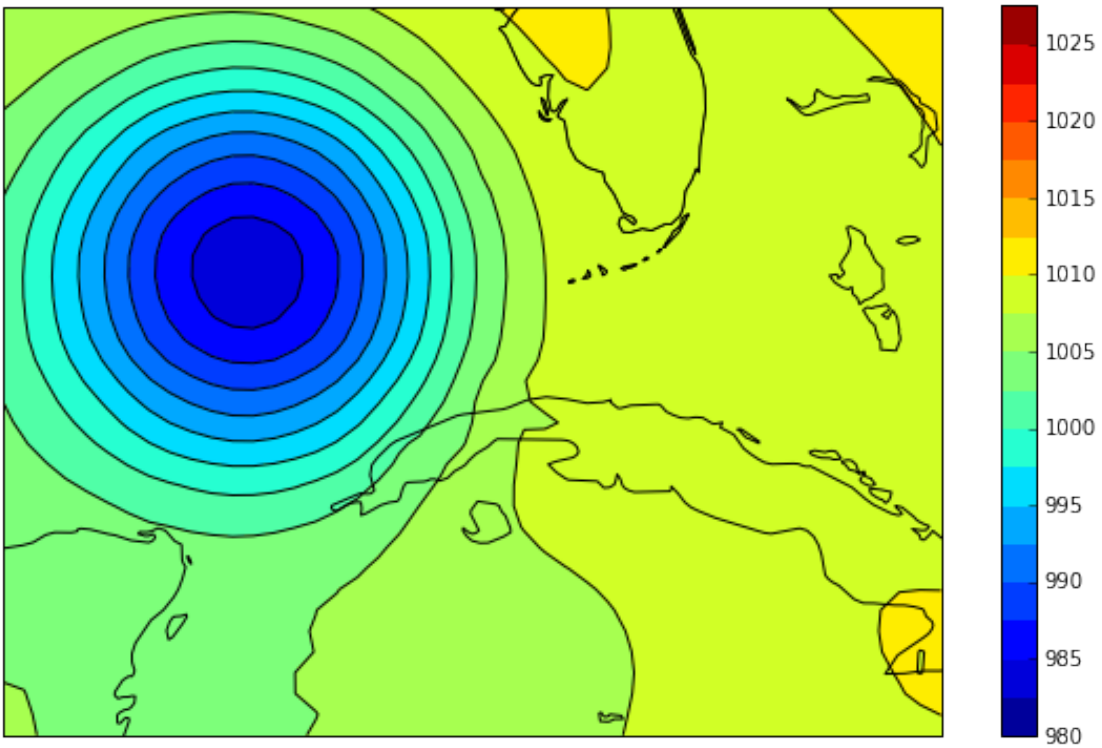
15.12 Your Turn!

15.12.1 Example 4.6: Cropping by Setting the X,Y Extents

```
In [21]: import numpy
         from matplotlib import pyplot
         from matplotlib.cm import get_cmap
         from cartopy import crs
         from cartopy.feature import NaturalEarthFeature
         from netCDF4 import Dataset
         from wrf import getvar, to_np, get_cartopy, latlon_coords
         from wrf import xy_to_ll, cartopy_xlim, cartopy_ylim
         from wrf import CoordPair, GeoBounds

         file_path = single_wrf_file()
         wrf_file = Dataset(file_path)

         # Get the terrain height
         slp = getvar(wrf_file, "slp", timeidx=0)

         # Get the cartopy object and the lat,lon coords
         cart_proj = get_cartopy(slp)
         lats, lons = latlon_coords(slp)
```

```
# Create a figure and get the GetAxes object
fig = pyplot.figure(figsize=(10, 7.5))
geo_axes = pyplot.axes(projection=cart_proj)

# Download and add the states and coastlines
# See the cartopy documentation for more on this.
states = NaturalEarthFeature(category='cultural',
                             scale='50m',
                             facecolor='none',
                             name='admin_1_states_provinces_shp')
geo_axes.add_feature(states, linewidth=.5)
geo_axes.coastlines('50m', linewidth=0.8)

# Set the contour levels so that all plots match
levels = numpy.arange(980.,1030.,2.5)

# Make the contour lines and fill them.
pyplot.contour(to_np(lons), to_np(lats),
               to_np(slp), levels=levels, colors="black",
               transform=crs.PlateCarree())
pyplot.contourf(to_np(lons), to_np(lats),
                to_np(slp), levels=levels,
                transform=crs.PlateCarree(),
                cmap=get_cmap("jet"))

# Add a color bar. The shrink often needs to be set
# by trial and error.
pyplot.colorbar(ax=geo_axes, shrink=.83)

# Set up the x, y extents

# Determine the center of the domain in grid coordinates
slp_shape = slp.shape
start_y = 0
center_y = int(slp_shape[-2]/2.) - 1
center_x = int(slp_shape[-1]/2.) - 1
end_x = int(slp_shape[-1]) - 1

# Get the lats and lons for the start, center, and end points
# (Normally you would just set these yourself)
center_latlon = xy_to_ll(wrf_file,
                         [center_x, end_x],
                         [start_y, center_y])

start_lat = center_latlon[0,0]
end_lat = center_latlon[0,1]
start_lon = center_latlon[1,0]
```

```
end_lon = center_latlon[1,1]

# Set the extents
geo_bounds = GeoBounds(CoordPair(lat=start_lat, lon=start_lon),
                       CoordPair(lat=end_lat, lon=end_lon))
geo_axes.set_xlim(cartopy_xlim(slp, geobounds=geo_bounds))
geo_axes.set_ylim(cartopy_ylim(slp, geobounds=geo_bounds))

pyplot.show()
```



16 5.0 Advanced Examples

These examples are a significant jump in difficulty from the previous ones.

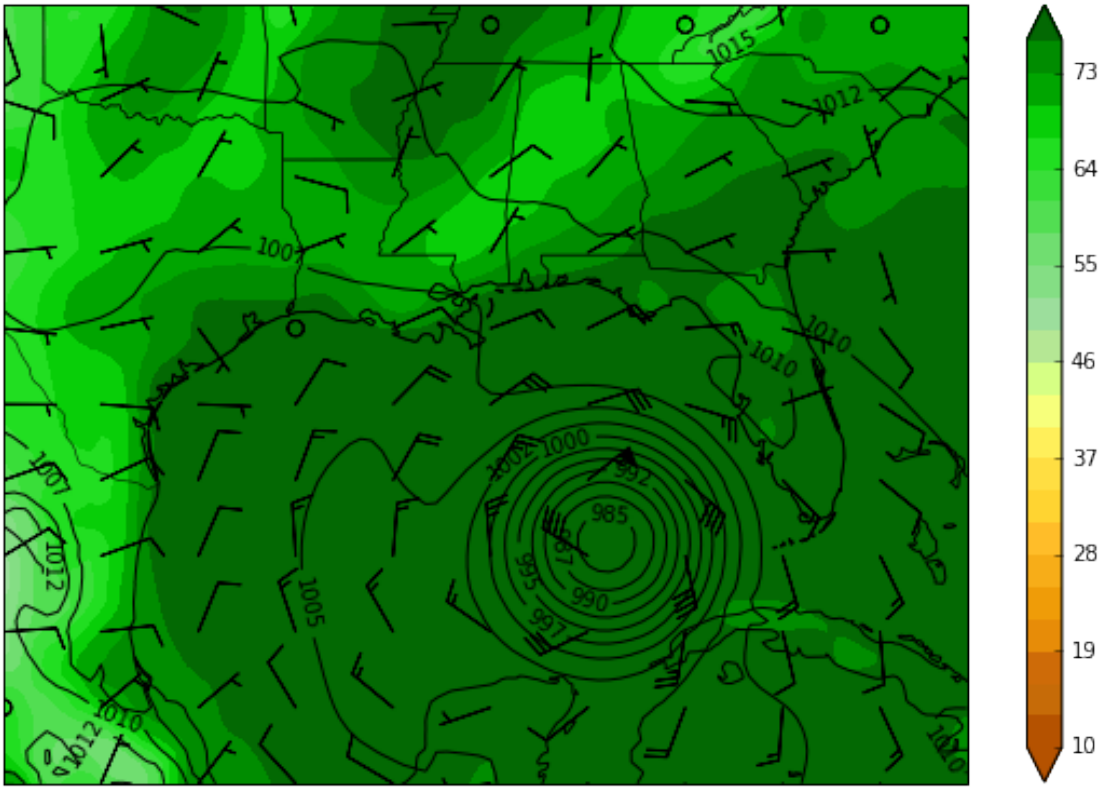
We won't have time to cover them in detail, so hang on to them for future reference when you begin making real world plots.

16.1 5.1: Overlaying Multiple Diagnostics

```
In [22]: import numpy
         from matplotlib import pyplot
         from matplotlib.cm import get_cmap
         from matplotlib.colors import from_levels_and_colors
```

```
from cartopy import crs
from cartopy.feature import NaturalEarthFeature
from netCDF4 import Dataset
from wrf import getvar, to_np, get_cartopy, latlon_coords, cartopy_xlim, cartopy_ylim

file_path = single_wrf_file()
wrf_file = Dataset(file_path)

# Get the slp, td2, u, and v variables
slp = getvar(wrf_file, "slp", timeidx=0)
td2 = getvar(wrf_file, "td2", timeidx=0, units="degF")
u_sfc = getvar(wrf_file, "ua", timeidx=0, units="kt")[0,:]
v_sfc = getvar(wrf_file, "va", timeidx=0, units="kt")[0,:]

# Get the cartopy object and the lat,lon coords
cart_proj = get_cartopy(slp)
lats, lons = latlon_coords(slp)

# Create a figure and get the GetAxes object
fig = pyplot.figure(figsize=(10, 7.5))
geo_axes = pyplot.axes(projection=cart_proj)

# Download and add the states and coastlines
# See the cartopy documentation for more on this.
states = NaturalEarthFeature(category='cultural',
                             scale='50m',
                             facecolor='none',
                             name='admin_1_states_provinces_shp')
geo_axes.add_feature(states, linewidth=.5)
geo_axes.coastlines('50m', linewidth=0.8)

# Manually setting the contour levels
slp_levels = numpy.arange(980.,1030.,2.5)
td2_levels = numpy.arange(10., 79., 3.)


# Manually setting the td2 RGB colors (normalized to 1)
td2_rgb = numpy.array([[181,82,0], [181,82,0],
                   [198,107,8], [206,107,8],
                   [231,140,8], [239,156,8],
                   [247,173,24], [255,189,41],
                   [255,212,49], [255,222,66],
                   [255,239,90], [247,255,123],
                   [214,255,132], [181,231,148],
                   [156,222,156], [132,222,132],
                   [112,222,112], [82,222,82],
                   [57,222,57], [33,222,33],
                   [8,206,8], [0,165,0],
```

```
                    [0,140,0], [3,105,3]]) / 255.0

td2_cmap, td2_norm = from_levels_and_colors(td2_levels, td2_rgb, extend="both")

# Make the contour lines and fill them.
slp_contours = pyplot.contour(to_np(lons),
                              to_np(lats),
                              to_np(slp),
                              levels=slp_levels,
                              colors="black",
                              transform=crs.PlateCarree())

pyplot.contourf(to_np(lons), to_np(lats),
                to_np(td2), levels=td2_levels,
                cmap=td2_cmap, norm=td2_norm,
                extend="both",
                transform=crs.PlateCarree())

# Plot the wind barbs, but only plot ~10 barbs in each direction.
thin = [int(x/10.) for x in lons.shape]
pyplot.barbs(to_np(lons[::thin[0], ::thin[1]]),
             to_np(lats[::thin[0], ::thin[1]]),
             to_np(u_sfc[::thin[0], ::thin[1]]),
             to_np(v_sfc[::thin[0], ::thin[1]]),
             transform=crs.PlateCarree())

# Add contour labels for pressure
pyplot.clabel(slp_contours, fmt="%i")

# Add a color bar. The shrink often needs to be set
# by trial and error.
pyplot.colorbar(ax=geo_axes, shrink=.86, extend="both")

# Set the map bounds
pyplot.xlim(cartopy_xlim(slp))
pyplot.ylim(cartopy_ylim(slp))

pyplot.show()
```

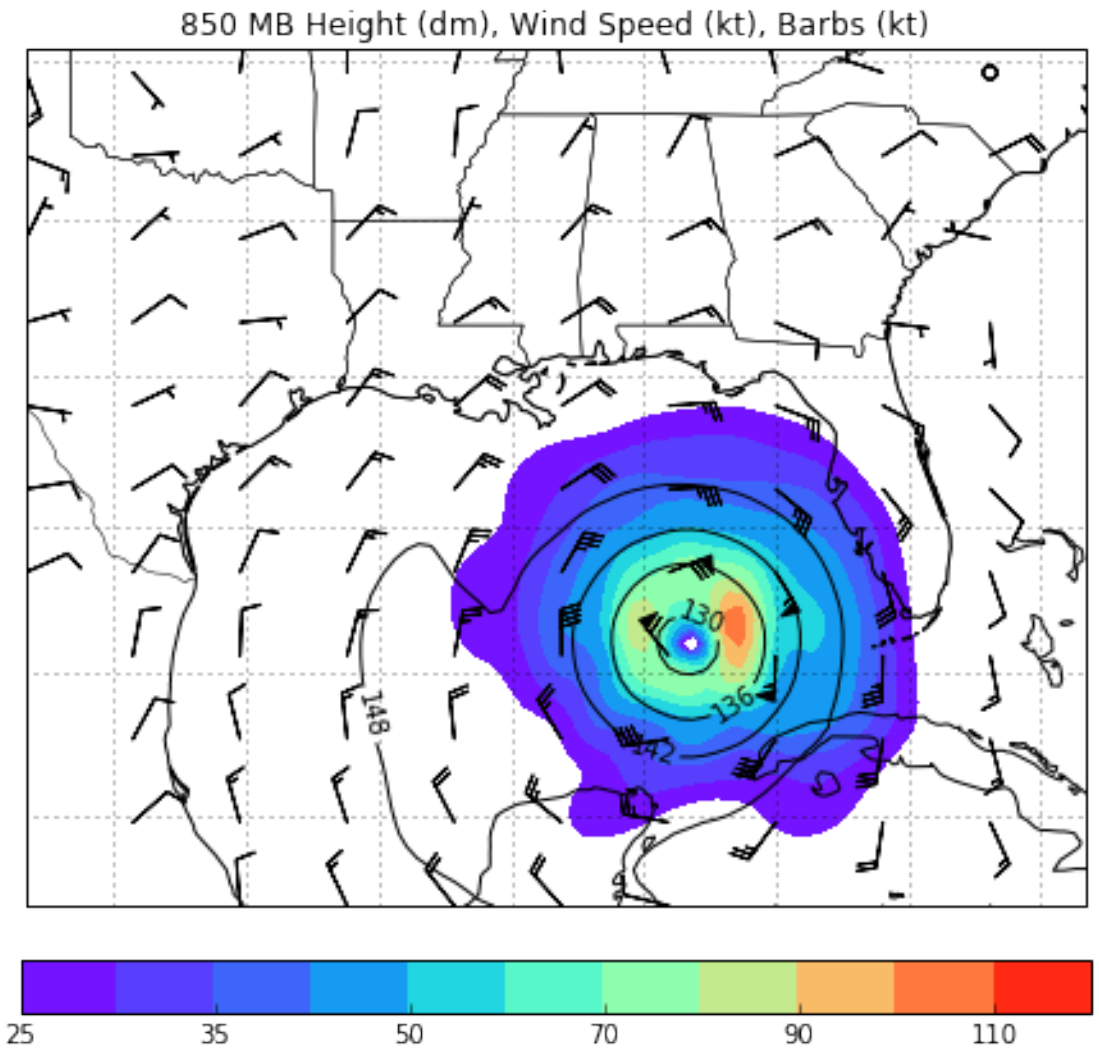


16.2 Example 5.2: 850 hPa Heights and Winds with *interplevel*

```
In [23]: import numpy
         from matplotlib import pyplot
         from matplotlib.cm import get_cmap
         from matplotlib.colors import from_levels_and_colors
         from cartopy import crs
         from cartopy.feature import NaturalEarthFeature
         from netCDF4 import Dataset
         from wrf import (getvar, to_np, get_cartopy, latlon_coords, interplevel,
                          cartopy_xlim, cartopy_ylim)

         file_path = single_wrf_file()
         wrf_file = Dataset(file_path)

         # Extract the pressure, geopotential height, and wind variables
         p = getvar(wrf_file, "pressure")
         z = getvar(wrf_file, "z", units="dm")
         ua = getvar(wrf_file, "ua", units="kt")
         va = getvar(wrf_file, "va", units="kt")
         wspd = getvar(wrf_file, "wspd_wdir", units="kt")[0,:]
```

```
# Interpolate geopotential height, u, and v winds to 850 hPa
ht_850 = interplevel(z, p, 850)
u_850 = interplevel(ua, p, 850)
v_850 = interplevel(va, p, 850)
wspd_850 = interplevel(wspd, p, 850)

# Get the lat/lon coordinates
lats, lons = latlon_coords(ht_850)

# Get the map projection information
cart_proj = get_cartopy(ht_850)

# Create the figure
fig = pyplot.figure(figsize=(10,7.5))
ax = pyplot.axes(projection=cart_proj)

# Download and add the states and coastlines
states = NaturalEarthFeature(category='cultural',
                             scale='50m',
                             facecolor='none',
                             name='admin_1_states_provinces_shp')
ax.add_feature(states, linewidth=0.5)
ax.coastlines('50m', linewidth=0.8)

# Add the 850 hPa geopotential height contours
levels = numpy.arange(130., 170., 6.)
contours = pyplot.contour(to_np(lons),
                          to_np(lats),
                          to_np(ht_850),
                          levels=levels,
                          colors="black",
                          transform=crs.PlateCarree())

pyplot.clabel(contours, inline=1, fontsize=10, fmt="%i")

# Add the wind speed contours
levels = [25, 30, 35, 40, 50, 60, 70, 80, 90, 100, 110, 120]
wspd_contours = pyplot.contourf(to_np(lons),
                                to_np(lats),
                                to_np(wspd_850),
                                levels=levels,
                                cmap=get_cmap("rainbow"),
                                transform=crs.PlateCarree())

pyplot.colorbar(wspd_contours, ax=ax, orientation="horizontal",
                pad=.05, shrink=.75)

# Add the 850 hPa wind barbs, only plotting 10 barbs in each direction
```

```
# Also, skipping the border barbs.
thin = [int(x/10.) for x in lons.shape]
pyplot.barbs(to_np(lons[::thin[0], ::thin[1]]),
          to_np(lats[::thin[0], ::thin[1]]),
          to_np(u_850[::thin[0], ::thin[1]]),
          to_np(v_850[::thin[0], ::thin[1]]),
          length=6,
          transform=crs.PlateCarree())

# Set the map bounds
ax.set_xlim(cartopy_xlim(ht_850))
ax.set_ylim(cartopy_ylim(ht_850))

ax.gridlines()

pyplot.title("850 MB Height (dm), Wind Speed (kt), Barbs (kt)")

pyplot.show()
```



16.3 Example 5.3: Cross Section

Begin by defining a cross section line in latitude, longitude coordinates.

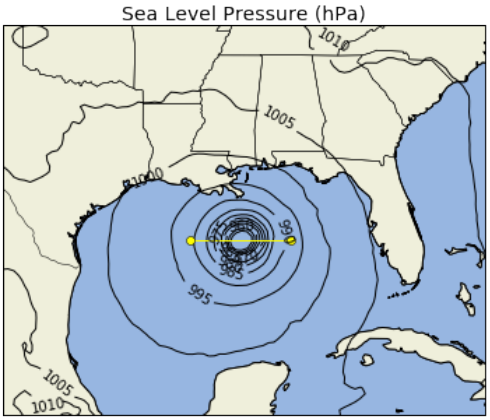
```
In [24]: from wrf import CoordPair

         cross_start = CoordPair(lat=27.1, lon=-91.7)
         cross_end = CoordPair(lat=27.1, lon=-86.7)
```

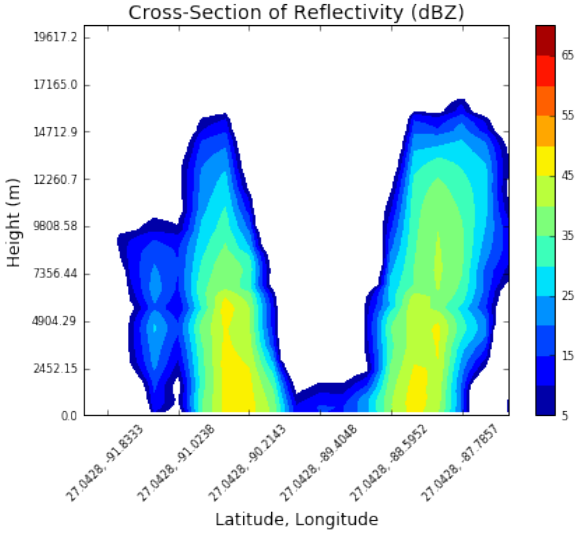
```
In [25]: import numpy
         from matplotlib import pyplot
         from matplotlib.cm import get_cmap
         from matplotlib.colors import from_levels_and_colors
         from cartopy import crs
         from cartopy.feature import NaturalEarthFeature, COLORS
```

```
from netCDF4 import Dataset
from wrf import (getvar, to_np, get_cartopy, latlon_coords, vertcross,
                 cartopy_xlim, cartopy_ylim)

file_path = multiple_wrf_files()
wrf_file = [Dataset(x) for x in file_path]

# Get the WRF variables
slp = getvar(wrf_file, "slp", timeidx=-1)
z = getvar(wrf_file, "z", timeidx=-1)
dbz = getvar(wrf_file, "dbz", timeidx=-1)
Z = 10**(dbz/10.) # Use linear Z for interpolation

# Compute the vertical cross-section interpolation.  Also, include the lat/lon
# points along the cross-section in the metadata by setting latlon to True.
z_cross = vertcross(Z, z, wrfin=wrf_file,
                    start_point=cross_start,
                    end_point=cross_end,
                    latlon=True, meta=True)

# Convert back to dBz after interpolation
dbz_cross = 10.0 * numpy.log10(z_cross)

# Get the lat/lon points
lats, lons = latlon_coords(slp)

# Get the cartopy projection object
cart_proj = get_cartopy(slp)

# Create a figure that will have 2 subplots (1 row, 2 columns)
fig = pyplot.figure(figsize=(15,5))
ax_slp = fig.add_subplot(1,2,1,projection=cart_proj)
ax_dbz = fig.add_subplot(1,2,2)

# Download and create the states, land, and oceans using cartopy features
states = NaturalEarthFeature(category='cultural', scale='50m', facecolor='none',
                                        name='admin_1_states_provinces_shp')
land = NaturalEarthFeature(category='physical', name='land', scale='50m',
                                      facecolor=COLORS['land'])
ocean = NaturalEarthFeature(category='physical', name='ocean', scale='50m',
                                       facecolor=COLORS['water'])

# Make the pressure contours
slp_levels = numpy.arange(950.,1030.,5)
slp_contours = ax_slp.contour(to_np(lons),
                    to_np(lats),
                    to_np(slp),
                    levels=slp_levels,
```

```
                    colors="black",
                    zorder=3,
                    transform=crs.PlateCarree())
# Add contour labels for pressure
ax_slp.clabel(slp_contours, fmt="%i")

# Draw the cross section line
ax_slp.plot([cross_start.lon, cross_end.lon],
            [cross_start.lat, cross_end.lat],
            color="yellow",
            marker="o",
            zorder=3,
            transform=crs.PlateCarree())

# Draw the oceans, land, and states
ax_slp.add_feature(ocean)
ax_slp.add_feature(land)
ax_slp.add_feature(states, linewidth=.5, edgecolor="black")

# Make the contour plot for dbz
dbz_levels = numpy.arange(5.,75.,5.)
dbz_contours = ax_dbz.contourf(to_np(dbz_cross), levels=dbz_levels, cmap=get_cmap("je
cb_dbz = fig.colorbar(dbz_contours, ax=ax_dbz)
cb_dbz.ax.tick_params(labelsize=8)

# Set the x-ticks to use latitude and longitude labels
coord_pairs = to_np(dbz_cross.coords["xy_loc"])
x_ticks = numpy.arange(coord_pairs.shape[0])
x_labels = [pair.latlon_str() for pair in to_np(coord_pairs)]
# Only keeping ~5 xticks
thin = [int(x/5.) for x in x_ticks.shape]
ax_dbz.set_xticks(x_ticks[1::thin[0]])
ax_dbz.set_xticklabels(x_labels[::thin[0]], rotation=45, fontsize=8)

# Set the y-ticks to be height
vert_vals = to_np(dbz_cross.coords["vertical"])
v_ticks = numpy.arange(vert_vals.shape[0])
# Only keeping ~8 vertical ticks
thin = [int(x/8.) for x in v_ticks.shape]
ax_dbz.set_yticks(v_ticks[::thin[0]])
ax_dbz.set_yticklabels(vert_vals[::thin[0]], fontsize=8)

# Set the x-axis and  y-axis labels
ax_dbz.set_xlabel("Latitude, Longitude", fontsize=12)
ax_dbz.set_ylabel("Height (m)", fontsize=12)

# Add a title
ax_slp.set_title("Sea Level Pressure (hPa)", {"fontsize" : 14})
```

```
ax_dbz.set_title("Cross-Section of Reflectivity (dBZ)", {"fontsize" : 14})

pyplot.show()
```





16.4 Example 5.4: Animations

```
In [26]: import numpy
         from matplotlib import pyplot, rc
         from matplotlib.cm import get_cmap
         from matplotlib.colors import from_levels_and_colors
         from matplotlib.animation import FuncAnimation
         from IPython.display import HTML
         from cartopy import crs
         from cartopy.feature import NaturalEarthFeature, COLORS
         from netCDF4 import Dataset
         from wrf import (getvar, to_np, get_cartopy, latlon_coords, vertcross,
                          cartopy_xlim, cartopy_ylim, ALL_TIMES)

         file_path = multiple_wrf_files()
         wrf_file = [Dataset(x) for x in file_path]

         # Get SLP for all times
         slp_all = getvar(wrf_file, "slp", timeidx=ALL_TIMES)

         # Get the cartopy projection object
         cart_proj = get_cartopy(slp_all)

         fig = pyplot.figure(figsize=(10,7.5))
         ax_slp = pyplot.axes(projection=cart_proj)
```

```
        # Download and create the states, land, and oceans using cartopy features
        states = NaturalEarthFeature(category='cultural', scale='50m', facecolor='none',
                                                name='admin_1_states_provinces_shp')
        land = NaturalEarthFeature(category='physical', name='land', scale='50m',
                                               facecolor=COLORS['land'])
        ocean = NaturalEarthFeature(category='physical', name='ocean', scale='50m',
                                                facecolor=COLORS['water'])

        slp_levels = numpy.arange(950.,1030.,5.)

        num_frames = slp_all.shape[0]

        def animate(i):
            ax_slp.clear()
            slp = slp_all[i,:]

            # Get the lat/lon coordinates
            lats, lons = latlon_coords(slp)

            ax_slp.add_feature(ocean)
            ax_slp.add_feature(land)
            ax_slp.add_feature(states, linewidth=.5, edgecolor="black")

            slp_contours = ax_slp.contour(to_np(lons),
                                          to_np(lats),
                                          to_np(slp),
                                          levels=slp_levels,
                                          colors="black",
                                          zorder=5,
                                          transform=crs.PlateCarree())

            # Add contour labels for pressure
            ax_slp.clabel(slp_contours, fmt="%i")

            # Set the map bounds
            ax_slp.set_xlim(cartopy_xlim(slp))
            ax_slp.set_ylim(cartopy_ylim(slp))

            return ax_slp

        ani = FuncAnimation(fig, animate, num_frames, interval=500, repeat_delay=1000, blit=Fa

        HTML(ani.to_html5_video())
```

Out[26]: <IPython.core.display.HTML object>