General Overview

-When doing research on syntopic weather systems, it is important to see their virtical structure by taking crossections with respesct to height levels. The purpose of this notebook is to explore a method to scan an area by vertical cross sections and to either draw a plot or returning a video (Essentially generated by Matplotlib.FuncAnimation()).

-Packages involved: xarray, matplotlib, cartopy, numpy, and metpy.
-Other than one interactive function, the notebook has all parameters for function set up to run. However, there were tons of work done for standardization, so feel free to change the variable you want to look at after you walk through this notebook.
-You don't really need to type any code, this is just for let you get familar with the funtions in this package!

In [ ]:
```
import xarray as xr, matplotlib.pyplot as plt, numpy as np
import metpy.calc as mpcalc
import matplotlib.colors as mplc
from metpy.units import units
import Scanner
```

The data we use here is from ERA5 dataset. Theoretically, the data can generate crossections from any dataset retrieved from ERA5 dataset as long as the variables in the dataset includes temperature and geopotential height.

-At first, we need to read the data by xarray function.
-Because the default names in the dataset is too long, we should change some variables/coordinate's name at first.
(You don't need to do this step at all if you are comfortable with the default!)
-Potential temperature and vorticity will be used later. Because we don't have them in the raw dataset, we need to calculate them by metpy function (mpcalc.potential_temperature()).

In [ ]:
```
ds=xr.open_dataset('./testdata/feb1512z.grib', engine='cfgrib')
ds=ds.metpy.quantify()
ds=ds.rename({"isobaricInhPa":"pressure","latitude":"lat", "longitude":"lon"})
 #You need to process some data at first before puting it into function
ds['thta']=mpcalc.potential_temperature(ds.pressure, ds.t) #We don't have thta
ds['vo'] = mpcalc.vorticity(ds.u,ds.v)
ds['z']=ds.z/9.8 #To calculate geopotential height, the value need to be devid
#Take a look with the processed dataset!
ds
```

```
Ignoring index file './testdata/feb1512z.grib.923a8.idx' incompatible with GRI
B file
/home/wen/anaconda3/envs/plot/lib/python3.10/site-packages/metpy/xarray.py:35
5: UserWarning: More than one time coordinate present for variable "u".
  warnings.warn('More than one ' + axis + ' coordinate present for variable'
```

Out[ ]: xarray.Dataset

▸ Dimensions: (**pressure**: 27, **lat**: 121, **lon**: 281)

▾ Coordinates:

number	()	int64	0
time	()	datetime64[ns]	2023-02-15T12:00:00
step	()	timedelta64[ns]	00:00:00
pressure	(pressure)	float64	1e+03 975.0 950.0 ... 12…
lat	(lat)	float64	50.0 49.75 49.5 ... 20.5 2…
lon	(lon)	float64	-140.0 -139.8 ... -70.25 -…
valid_time	()	datetime64[ns]	2023-02-15T12:00:00

▾ Data variables:

z	(pressure, lat, lon)	float32	<Quantity([[[91.4202 90....
pv	(pressure, lat, lon)	float32	<Quantity([[[-1.2834789e…
t	(pressure, lat, lon)	float32	<Quantity([[[278.68237 2…
u	(pressure, lat, lon)	float32	<Quantity([[[9.163666 9.…
v	(pressure, lat, lon)	float32	<Quantity([[[-2.067749 -2…
w	(pressure, lat, lon)	float32	<Quantity([[[-2.88496017…
thta	(pressure, lat, lon)	float64	<Quantity([[[278.682373…
vo	(pressure, lat, lon)	float64	<Quantity([[[-2.91261154…

▸ Indexes: (3)

▾ Attributes:

GRIB_edition :	1
GRIB_centre :	ecmf
GRIB_centreDe…	European Centre for Medium-Range Weather Forecasts
GRIB_subCentre :	0
Conventions :	CF-1.7
institution :	European Centre for Medium-Range Weather Forecasts
history :	2024-01-24T17:04 GRIB to CDM+CF via cfgrib-0.9.10.3/ecCodes-2.27.1 with {"source": "testdata/feb1512z.grib", "filter_by_keys": {}, "encode_cf": ["parameter", "time", "geography", "vertical"]}

There are many occasions that we only need to look at a small area do out research and draw crossections, so the following `selection()` function is defined.
-The function acts as a specified version of .sel() function in xarray

What you can specify in this function:
-Only the veriables you want for later use.
-Area of the interest
-Pressure level of interest (Only one level at a time! Essential for the initial map, but please

don't specify when doing crossection!)

-The index value of time you want (Only one at a time!)

In [ ]:
```
def selection(dataset,  vrbs, extent=None, plevel=None, tidx=None):
    r'''
    Parameters
    ----------
    dataset: 'xarray.Dataset'
        The dataset imported before.
    vrbs: 'list'
        Variables want to keep in the new xarray dataset. Should be a list of r
        Example:['t', 'z', 'w']
    extent=None: 'list'
        Should be 2 pairs of lat, lon values in a least, use the dataset's spar
        (Example: [-130,-60,20, 52] (This is a CONUS view!))
    plevel:'int'
        The pressure level want to keep in the new dataset.
        (One level only)
    tidx:'int'
        Time index in the old dataset that you want to keep in the new dataset
        (One time only)

    Returns
    -------
    'xarray.Dataset'
        The filtered dataset.
    '''
```

Since we haven't have our points for cross sections defined, we need to prepare a dataset on some pressure level to identify our area of interest. Hence, we need to slice out some data in one pressure level to generates maps in later functions.

-Note: Certainly, we can't make cross section for a dataset that only in one pressure level, so we will call `selection()` at a later time!

In [ ]:
```
ds1=Scanner.selection(ds, ['z', 'thta', 'vo'], extent=[-130,-60,20, 52], pleve
ds1
```

Out[ ]: xarray.Dataset

▸ Dimensions: (**lat**: 121, **lon**: 241)

▾ Coordinates:

number	()	int64	0
time	()	datetime64[ns]	2023-02-15T12:00:00
step	()	timedelta64[ns]	00:00:00
pressure	()	float64	500.0
lat	(lat)	float64	50.0 49.75 49.5 ... 20.5 20.25 20.0
lon	(lon)	float64	-130.0 -129.8 ... -70.25 -70.0
valid_time	()	datetime64[ns]	2023-02-15T12:00:00

▾ Data variables:

z	(lat, lon)	float32	<Quantity([[5567.1235 5568.144 ...
thta	(lat, lon)	float64	<Quantity([[304.59471262 304.580...
vo	(lat, lon)	float64	<Quantity([[-6.56933477e-05 -5.8...

▸ Indexes: (2)

▾ Attributes:

GRIB_edition :	1
GRIB_centre :	ecmf
GRIB_centreDe…	European Centre for Medium-Range Weather Forecasts
GRIB_subCentre :	0
Conventions :	CF-1.7
institution :	European Centre for Medium-Range Weather Forecasts
history :	2024-01-24T14:00 GRIB to CDM+CF via cfgrib-0.9.10.3/ecCodes-2.27.1 with {"source": "testdata/feb1512z.grib", "filter_by_keys": {}, "encode_cf": ["parameter", "time", "geography", "vertical"]}

The next step is to confirm the coordinate that we wnat to do the scanning.

However, before introducing the function, I need to introduce the parameters for setting the plotting parameters at first.
-All the specifing parameters for plotting in any plot-related functions below should be typed in a dictionary at first.

The following is the default dictionary in the package!:

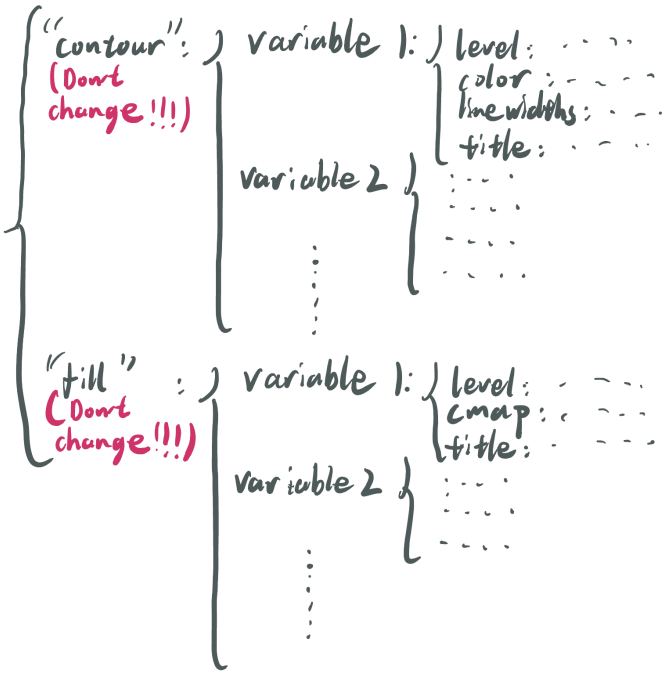
In [ ]:
```
###Example!

pv_cmap = list(reversed(['#f80000', '#fa2c0c', '#fc4017', '#fe5121', '#ff5f2c'
                '#ff864a', '#ff9153', '#ff9c5d', '#ffa666', '#ffb070', '#ffba79
                '#d5b17c', '#b29e73', '#968965', '#867152', '#845336', '#9b0000
w_cmap = list(reversed(['#00007f', '#2f278e', '#49469c', '#5f65aa', '#7485b8',
pv_cmap = mplc.ListedColormap(pv_cmap)
```

```
w_cmap = mplc.ListedColormap(w_cmap)

default={'contour':{
                 'thta':{'level': np.arange(250, 450, 3),
                         'color':'red',
                         'linewidths':1,
                         'title':"Potential temperature (K)"},
                    'z':{'level': np.arange(0, 10000, 60),
                         'color':'black',
                         'linewidths':1,
                         'title': "Geopotential height (m)"},
                    't':{'level': np.arange(0, 400, 3),
                         'color':'black',
                         'linewidths':1,
                         'title': "Temperature (K)"}},
            'fill':{
                  'vo':{'level': np.arange(5e-5,40e-5,5e-5),
                         'cmap':plt.cm.YlOrRd,
                         'title': "Relative vorticity(1/s)"},
                   'pv':{'level': np.arange(-1e-6,9.1e-6,0.5e-6),
                         'cmap':pv_cmap, #Defined in the .py file. You are wel
                         'title': "Potential vorticity(PVU)"},
                   'w':{'level': np.arange(-4, 4.1, 0.5),
                         'cmap':w_cmap, #Defined in the .py file. You are welc
                         'title': "Omega(Pa/s)"}}}
```

As you can see, there are in total of 3 layers in this dictionary. A picture for explaining dictionary structure is mentioned in the README file, but I am also putting one here for quick reference.



Please feel free to change the plotfile dictionary file of you want to plot some other parameters, but it has to be in the following format:
If it is a contour, attach the following to `'contour'`:

```
'variable name':#Your variable name in the dataset
            {
             'level': #Your contour interval, should be a numpy array
             'color': #Your contour color, should be a string with
full name of color (It will be written on the title!)
             'linewidths': #How wide you want your line is. should be
in int type
             'title': #Your variable's official name with a unit
(Named as title because it is mainly used in title
            }
```

If it is a filled contour, attach the following to `'fill'`:

```
'variable name': #Your variable name in the dataset
           {
            'level': #Your contour interval, should be a numpy array
            'cmap': #Your colormap, should be the name of the
colorbar after plt.cm
            'title': #Your variable's official name with a unit
(Named as title because it is mainly used in title
            }
```

Here is a link for colormaps in matplotlib

It doesn't need to be a perfect match for the name of variable to do the plotting--The plotting fuction actually only runs when the **same variable name** is in both the dataset and the plotfile dictionary here.

Now I can start to explain the function `estimation(dataset, locations = [], width = 14, steps = 25, plotfile='default')` This function is used to estimate the coordinate of the points you want to do the cross section.
So...Yes, this is an interactive function. You should follow the prompt to get the information for `Scanner()` function later.
Regarding the parameters:

> "dataset" is the dataset you want to do the preview on. **Remember: the dataset should have only one vertical level here**
>
> "locations" is the list of coordinates you want to do the cross section, and it should be in the format of [lat1, lon1, lat2, lon2, ...]
>
> "width" is the width of the cross section you want to do. The unit is in degree.
>
> "steps" is the number of slices you want to do. The default is 25. **This is not related to "steps" in ```Scanner()``` and only used as referencing purpose**
>
> "plotfile" is the dictionary you want to use for plotting, default is a string 'default'

After you put your desired coordinates, you would see a like and three or more colored points on the plot, they may be interpreted as following:

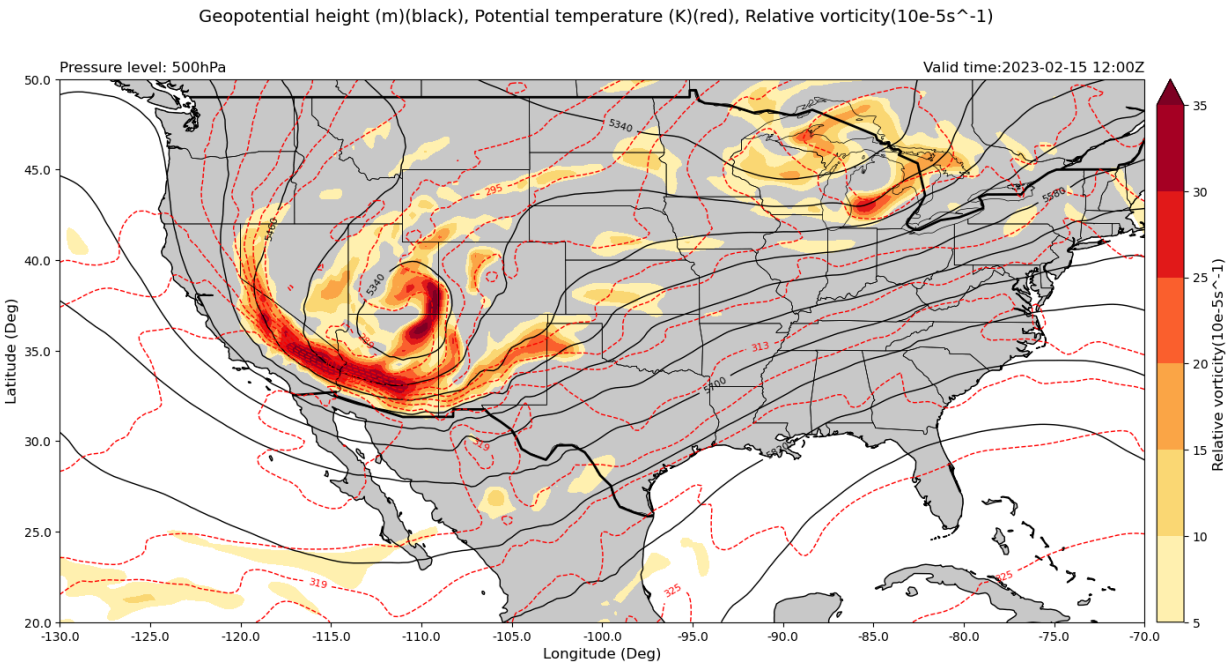
-pos1(Show later as red dot): Initial point

-pos2(Show later as orange dot): Critical points in the middle that determine the curvature of the line

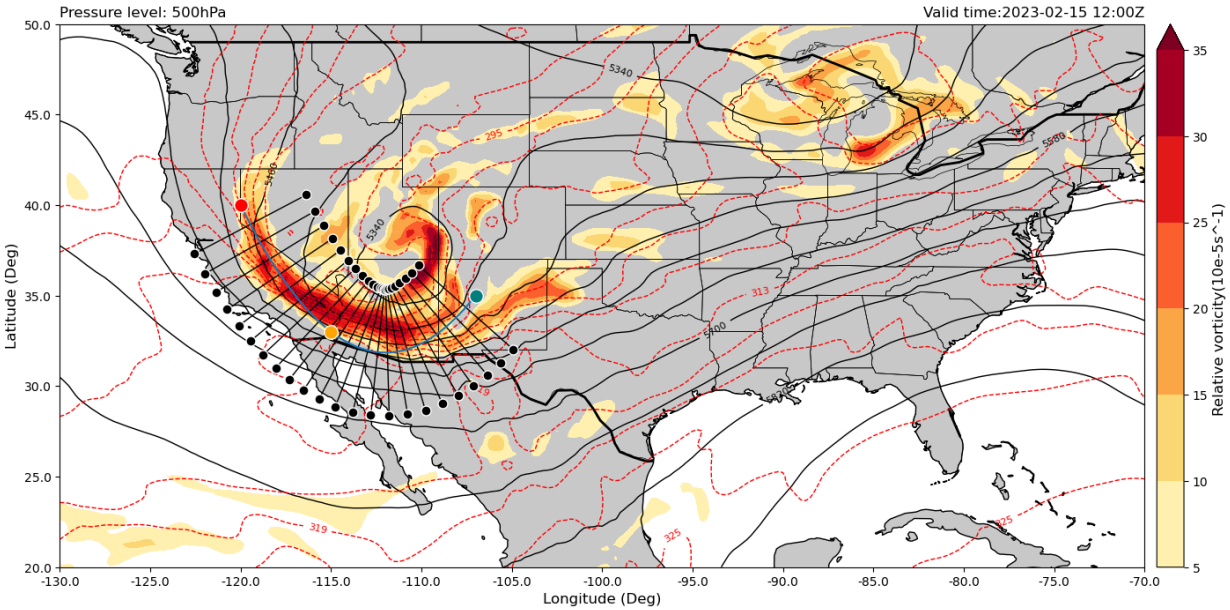
-pos3(Show later as teal dot): Last point

Here is a set of coordinates for demonstration of the functionality: 40.0, -120.0, 33, -115, 35.0, -107.0

In [ ]:
```
#Example coordinates: 40.0, -120.0, 33, -115,  35.0, -107.0
outcoords1, map1 = Scanner.estimation(ds1)
```




```
Current: (lat/lon1, lat/lon2)(40.0/-120.0, 35.0/-107.0)
<Figure size 640x480 with 0 Axes>
```




```
Output: (lat/lon1, lat/lon2)(40.0/-120.0, 35.0/-107.0)
```

In [ ]:
```
#We can check the locations of all the black points
outcoords1
```

Out[ ]:
```
[[[40.56720877111114, -116.36214138703303],
  [39.67411026892688, -115.87373104922801],
  [38.86561151910638, -115.39565212597843],
  [38.14281249847767, -114.93044631636263],
  [37.50674656697427, -114.48135004735805],
  [36.95820737773084, -114.05245804693712],
  [36.497452138413195, -113.6488766837365],
  [36.12372425962451, -113.27680130094068],
  [35.83454399880537, -112.94338466946186],
  [35.62477443098783, -112.65616842020097],
  [35.485641005934106, -112.42176813544339],
  [35.404209621710706, -112.24357322241896],
  [35.364198663652544, -112.11868655419035],
  [35.348952096691754, -112.03529715799084],
  [35.34629632739979, -111.9725582329275],
  [35.35309394527989, -111.90442194736042],
  [35.376543747501415, -111.80624557215987],
  [35.43127329904117, -111.66061723169312],
  [35.53429375285964, -111.45954764988974],
  [35.70075354767177, -111.2030711276934],
  [35.94189944852265, -110.89622592989383],
  [36.264925627893966, -110.54622241672831],
  [36.67376860939743, -110.16055349125628]],
 [[37.30310372888886, -122.55452527963362],
  [36.21547306440645, -121.95960228410533],
  [35.19220098089362, -121.35434787402157],
  [34.23218750152233, -120.73622035030402],
  [33.33439926635907, -120.1019832859753],
  [32.498042622269146, -119.44754195306288],
  [31.722860361586797, -118.76778998293015],
  [31.00960907370883, -118.05653203239267],
  [30.360768501194617, -117.30661533053814],
  [29.78147556901216, -116.51049824646569],
  [29.28050482739922, -115.66156519788996],
  [28.870790378289286, -114.75642677758104],
  [28.568613836347446, -113.79798011247631],
  [28.390631236641575, -112.7980361753425],
  [28.349016172600194, -111.7774417670725],
  [28.44690605472011, -110.76224471930624],
  [28.67710208583192, -109.77708776117348],
  [29.024976700958806, -108.83938276830688],
  [29.473518747140364, -107.95711901677691],
  [30.007579785661555, -107.13026220563994],
  [30.615913051477346, -106.35377407010617],
  [31.29132437210604, -105.62044424993834],
  [32.0298772239359, -104.92277984207706]]]
```

Now, here is the scanner function `scanner(slice_idx, dataset, coords_list, steps='default', plotfile=plotfile, plot=True)`, depending of the input, the scanner function has output in 3 types.

-Video

-Image
-Frame for video/gif (Should just be used by function itself to make video)

Among the parameters for the function, for generating output, you must have:
-dataset :Your dataset to do scanning
-coords_list :Your list of coordinates for scanning, generated from coords_generator()
-plotfile :As you can see when you check the function, although you must have it, you are free to use the default plotfile existing in this notebook (Just remember to name your dictionary as plotfile.

To have **video** as output, you need to do the following besides setting up above essential parameters:
-Set slice_idx = None: (Since you are generating a video instead of an image of a specific slice of cross section.)
-Set plot = False: (Setting the mod for video)
Example: `scanner(None, ds2, coords_list, plotfile=plotfile, plot=False)`

To have **image** as output, you need to do the following besides setting up above essential parameters:
-Set slice_idx as an integer: (indicating the index of cross section slice to be obtained)
-Set plot = True: (This is also a default, so you can skip this part)
Example: `scanner(0, ds2, coords_list, plotfile=plotfile)`

Regarding the steps, that is about how many crossections images (or the number of steps) you want to get for the scanning.
The default number of step is the magnitude in degrees plus 1 between the start and end point.

- If you want more steps, put integers in `steps`, like `steps=10`.

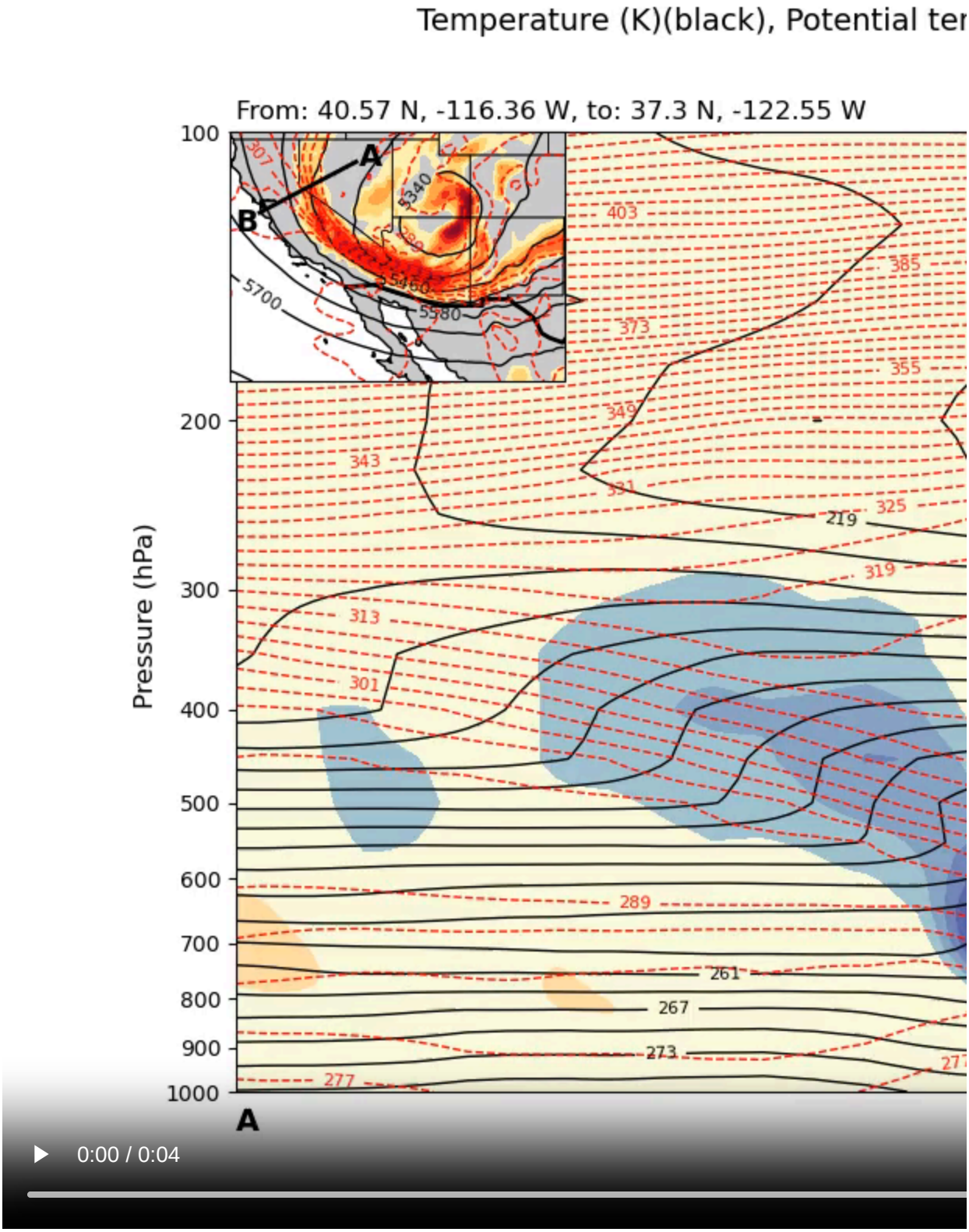
In order to see the cross section, put the second item in the output tuple to `prec=`.

Note: The core function for finding intermidiate points is realized by metpy.interpolate.geodesic()

Here is an example for generating crossection at the start point.

In [ ]:
```
ds3 = Scanner.selection(ds, ['t', 'thta', 'w'], extent=[-130,-60,20, 52])
Scanner.scanner(None, ds3, outcoords1, plot=False, prec=map1)
```

```
Currently slicing: (40.56720877111114, -116.36214138703303), (37.3031037288888
6, -122.55452527963362)
Currently slicing: (40.56720877111114, -116.36214138703303), (37.3031037288888
6, -122.55452527963362)
Currently slicing: (39.67411026892688, -115.87373104922801), (36.2154730644064
5, -121.95960228410533)
Currently slicing: (38.86561151910638, -115.39565212597843), (35.1922009808936
2, -121.35434787402157)
Currently slicing: (38.14281249847767, -114.93044631636263), (34.2321875015223
3, -120.73622035030402)
Currently slicing: (37.50674656697427, -114.48135004735805), (33.3343992663590
7, -120.1019832859753)
Currently slicing: (36.95820737773084, -114.05245804693712), (32.4980426222691
46, -119.44754195306288)
Currently slicing: (36.497452138413195, -113.6488766837365), (31.7228603615867
97, -118.76778998293015)
Currently slicing: (36.12372425962451, -113.27680130094068), (31.0096090737088
3, -118.05653203239267)
Currently slicing: (35.83454399880537, -112.94338466946186), (30.3607685011946
17, -117.30661533053814)
Currently slicing: (35.62477443098783, -112.65616842020097), (29.7814755690121
6, -116.51049824646569)
Currently slicing: (35.485641005934106, -112.42176813544339), (29.280504827399
22, -115.66156519788996)
Currently slicing: (35.404209621710706, -112.24357322241896), (28.870790378289
286, -114.75642677758104)
Currently slicing: (35.364198663652544, -112.11868655419035), (28.568613836347
446, -113.79798011247631)
Currently slicing: (35.348952096691754, -112.03529715799084), (28.390631236641
575, -112.7980361753425)
Currently slicing: (35.34629632739979, -111.9725582329275), (28.34901617260019
4, -111.7774417670725)
Currently slicing: (35.35309394527989, -111.90442194736042), (28.4469060547201
1, -110.76224471930624)
Currently slicing: (35.376543747501415, -111.80624557215987), (28.677102085831
92, -109.77708776117348)
Currently slicing: (35.43127329904117, -111.66061723169312), (29.0249767009588
06, -108.83938276830688)
Currently slicing: (35.53429375285964, -111.45954764988974), (29.4735187471403
64, -107.95711901677691)
Currently slicing: (35.70075354767177, -111.2030711276934), (30.00757978566155
5, -107.13026220563994)
Currently slicing: (35.94189944852265, -110.89622592989383), (30.6159130514773
46, -106.35377407010617)
Currently slicing: (36.264925627893966, -110.54622241672831), (31.291324372106
04, -105.62044424993834)
```



Old function, it is now renamed as `estimation_3pts()`

Now it's the time to explain the function ```estimation(dataset, pos1=[None, None], pos2=[None, None], pos3=[None, None], plotfile=plotfile)```.
This function is an interactive function for generating coordinate for the scanner. What you must put is a dataset with lat/lon or latitude/longitude as x&y axis and a plotfile for plotting

parameters. Initially, no matter have you decided your coordinates yet or not for the scanner, a base map is drawn for you to see what happens on the lat&lon surface. Lat and lon are given by ticks for adding convenience.

At the beginning, if you have your coordinates decided you can just put each pair of them in the location you want(It can also run if you have't decided one of the locations yet)

Note, each of the pos# slot corresponds to following:
-pos1(Show later as red dot): Initial point of first cross section
-pos2(Show later as orange dot): Final point of first cross section
-pos3(Show later as teal dot): First point of last cross section

If you want to fix your coordinate, type something other than **y** or **Y** (This is case insensitive!!!). Your new input should be numbers typed like following "lat1, lon1, lat2, lon2, lat3, lon3" (compatible for both float and int, mainly requiring each number separated by commas, no worries on the spaces)

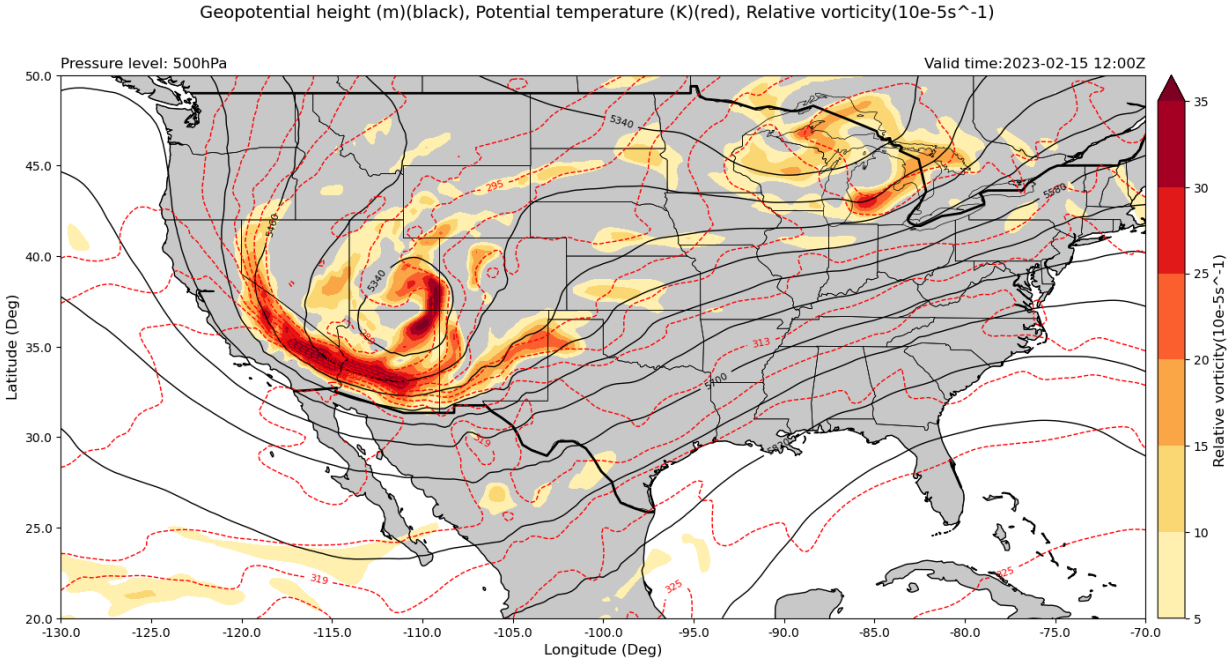
However, if you just want to change only one parameter, you can always type **None or something else for the rest of part**.
If you want to leave by whatever reason, just type **y** or **Y**.

A test lat/lon pairs is given here: 32.0, -125.0, 47.0, -120.0, 25.0, -102.0

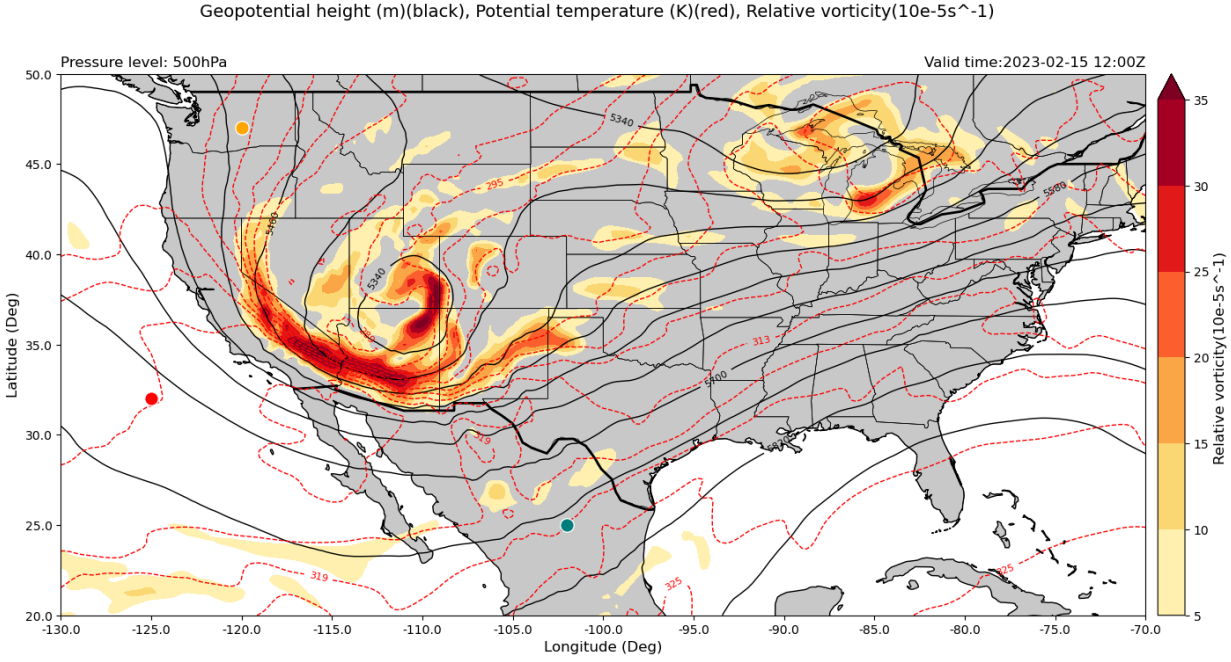
Remember you need to put the dataset and plotfile before you run the function! (For here as I have everything set up already so you can ignore)
And, you can assign an variable in front of the function to store the output! (I also let the function print the coords out so not necessary).

New feature! The output will now be a tuple. If you want to see the location of the cross-session in the scanner, you can use the second item in the output tuple in the `prec=` in `scanner()`

In [ ]:
```
#Eventually, the output in in following format [1st slice start (lat, lon), 1s
#You can assign the output in to one variable.
output=Scanner.estimation_3pts(ds1)
```




```
Current: (lat/lon1, lat/lon2, lat/lon3)(32.0, -125.0, 47.0, -120.0, 25.0, -10
2.0)
<Figure size 640x480 with 0 Axes>
```




```
Output: (lat/lon1, lat/lon2, lat/lon3)[32.0, -125.0, 47.0, -120.0, 25.0, -102.
0]
```

In [ ]:
```
#Also check if the output is assigned!
output[0]
```

Out[ ]:
```
(32.0, -125.0, 47.0, -120.0, 25.0, -102.0)
```

At this time, since we were choosing a certain pressure level, we need to run `selection()` again to get a dataset with info on **all pressure levels**.

-You can also assign other variables to your new dataset for scanning (For example, here chooses **temperature** and **vertial wind** this time instead of geopotential height and vorticity).

In [ ]:
```
#Do another selection (Please don't select a pressure level this time!)
ds2=Scanner.selection(ds, ['t', 'thta', 'w'], extent=[-130,-60,20, 52])
ds2
```

Out[ ]: xarray.Dataset

▸ Dimensions: (**pressure**: 27, **lat**: 121, **lon**: 241)

▾ Coordinates:

number	()	int64	0
time	()	datetime64[ns]	2023-02-15T12:00:00
step	()	timedelta64[ns]	00:00:00
pressure	(pressure)	float64	1e+03 975.0 950.0 ... 12…
lat	(lat)	float64	50.0 49.75 49.5 ... 20.5 2…
lon	(lon)	float64	-130.0 -129.8 ... -70.25 -…
valid_time	()	datetime64[ns]	2023-02-15T12:00:00

▾ Data variables:

t	(pressure, lat, lon)	float32	<Quantity([[[278.04175 2…
thta	(pressure, lat, lon)	float64	<Quantity([[[278.041748…
w	(pressure, lat, lon)	float32	<Quantity([[[-0.08622265…

▸ Indexes: (3)

▾ Attributes:

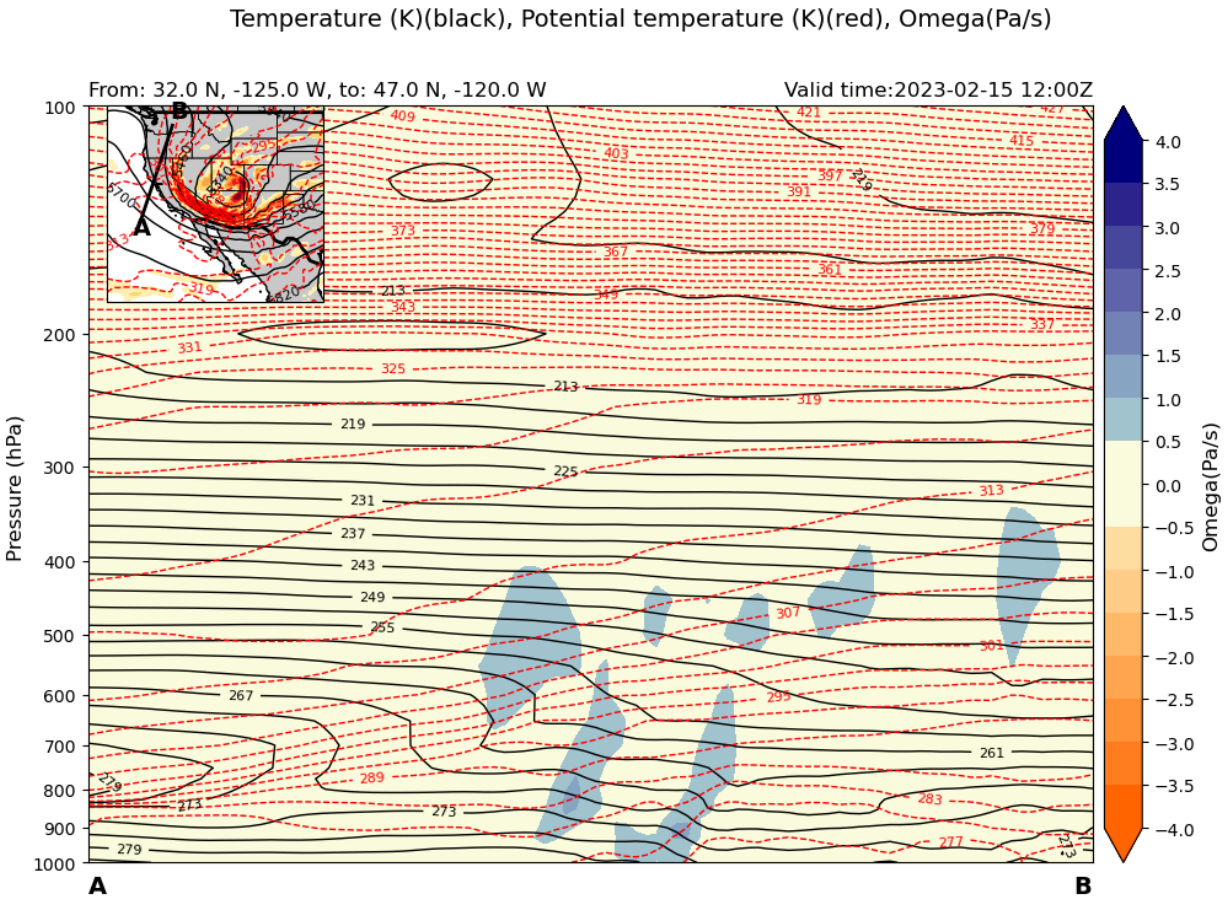
GRIB_edition :	1
GRIB_centre :	ecmf
GRIB_centreDe…	European Centre for Medium-Range Weather Forecasts
GRIB_subCentre :	0
Conventions :	CF-1.7
institution :	European Centre for Medium-Range Weather Forecasts
history :	2024-01-24T14:00 GRIB to CDM+CF via cfgrib-0.9.10.3/ecCodes-2.27.1 with {"source": "testdata/feb1512z.grib", "filter_by_keys": {}, "encode_cf": ["parameter", "time", "geography", "vertical"]}

`scanner_rect()` is the old version of the scanner function. The usage is similar. Cells below are the example:

In [ ]:
```
coordsinfo, map = output
Scanner.scanner_rect(0, ds2, coordsinfo, plotfile='default', plot=True, prec=ma
#As it is doing indexing, you can change the index to -1 for getting the last
```

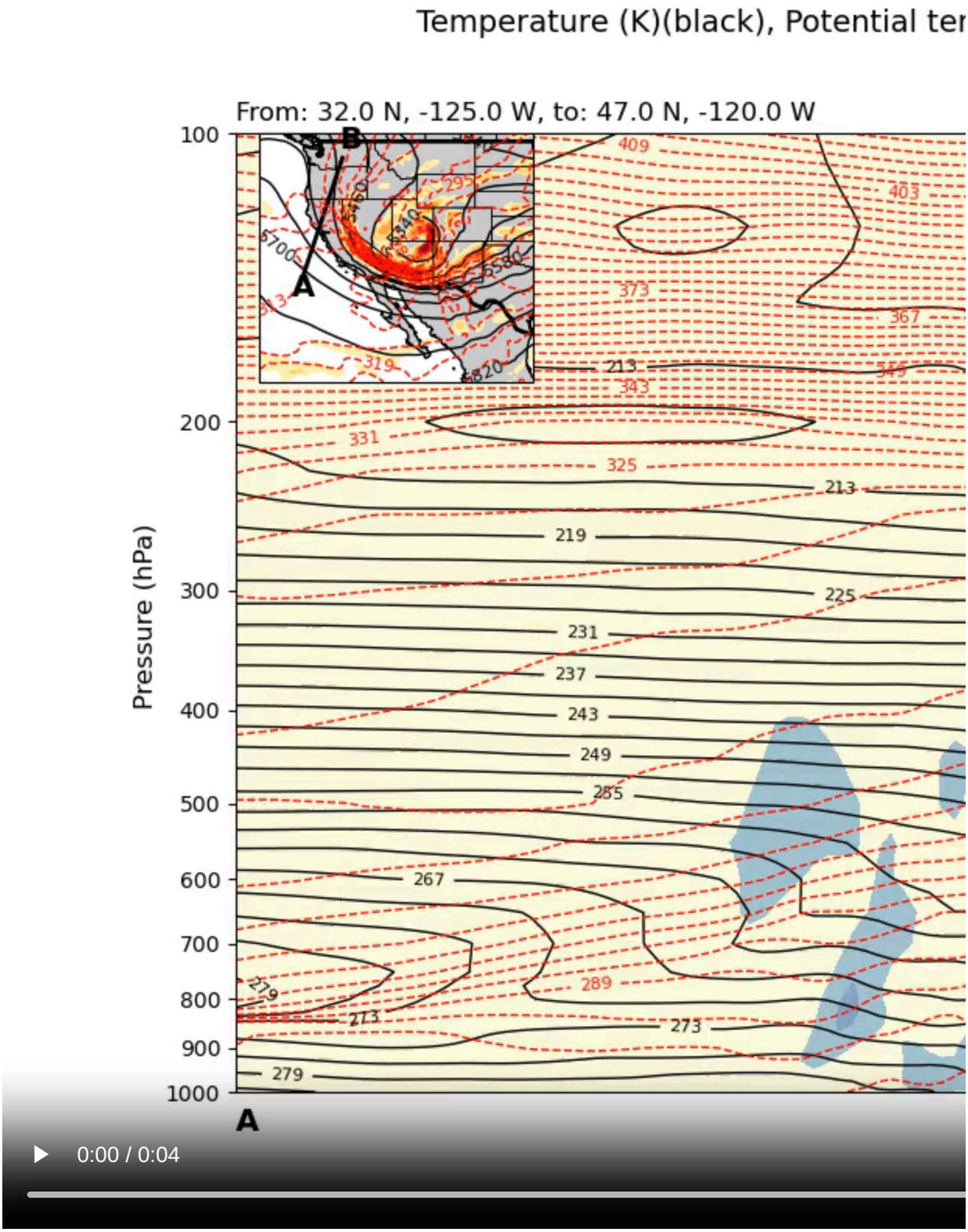
```
Currently slicing: (32.0, -125.00000000000001), (47.0, -120.00000000000001)
```

Out[ ]:
```
<Axes: title={'left': 'From: 32.0 N, -125.0 W, to: 47.0 N, -120.0 W', 'right':
'Valid time:2023-02-15 12:00Z'}, ylabel='Pressure (hPa) '>
<Figure size 640x480 with 0 Axes>
```



In [ ]:
```
#Here is an example for video (Takes a bit time ~30s on my own laptop!)
Scanner.scanner_rect(None, ds2, coordsinfo, plotfile='default', plot=False, pre
```

```
Currently slicing: (32.0, -125.00000000000001), (47.0, -120.00000000000001)
Currently slicing: (32.0, -125.00000000000001), (47.0, -120.00000000000001)
Currently slicing: (31.782745463993912, -123.9381921606786), (46.7827454639939
1, -118.9381921606786)
Currently slicing: (31.556672232133703, -122.88143789643911), (46.556672232133
71, -117.88143789643911)
Currently slicing: (31.321898956474932, -121.82986544825006), (46.321898956474
93, -116.82986544825006)
Currently slicing: (31.078547097300742, -120.78359321733072), (46.078547097300
74, -115.78359321733072)
Currently slicing: (30.82674070169304, -119.74272977580121), (45.8267407016930
4, -114.74272977580121)
Currently slicing: (30.566606184876512, -118.7073739086147), (45.5666061848765
1, -113.7073739086147)
Currently slicing: (30.298272115094118, -117.6776146849569), (45.2982721150941
2, -112.6776146849569)
Currently slicing: (30.021869002712048, -116.65353155720108), (45.021869002712
05, -111.65353155720108)
Currently slicing: (29.737529094189373, -115.63519448543497), (44.737529094189
37, -110.63519448543497)
Currently slicing: (29.44538617148482, -114.6226640855248), (44.4453861714848
2, -109.6226640855248)
Currently slicing: (29.14557535740953, -113.61599179865532), (44.1455753574095
3, -108.61599179865532)
Currently slicing: (28.838232927372253, -112.61522008027546), (43.838232927372
25, -107.61522008027546)
Currently slicing: (28.523496127901442, -111.62038260639073), (43.523496127901
44, -106.62038260639073)
Currently slicing: (28.20150300226906, -110.63150449517073), (43.2015030022690
64, -105.63150449517073)
Currently slicing: (27.872392223482706, -109.64860254188304), (42.872392223482
706, -104.64860254188304)
Currently slicing: (27.536302934857318, -108.67168546522115), (42.536302934857
32, -103.67168546522115)
Currently slicing: (27.193374598325054, -107.70075416316122), (42.193374598325
05, -102.70075416316122)
Currently slicing: (26.843746850591934, -106.73580197656042), (41.843746850591
934, -101.73580197656042)
Currently slicing: (26.48755936720376, -105.77681495879486), (41.4875593672037
6, -100.77681495879486)
Currently slicing: (26.12495173454016, -104.8237721498271), (41.1249517345401
6, -99.8237721498271)
Currently slicing: (25.756063329716277, -103.87664585318966), (40.756063329716
28, -98.87664585318966)
Currently slicing: (25.381033208335005, -102.93540191447215), (40.381033208335
005, -97.93540191447215)
```



As one of the final goal of this notebook is to make video by the scanner function. We can change some parameters for the scanner function to run the embedded matplotlib.animation.FuncAnimation() here.

Other than this set of functions, there are other functions that used to generate visualizations for other purposes.

These functions are:

`baseplot(dataset,plotfile = 'default', info = False, fig = None)` -- Basic function for plotting one image based on the dataset at one given time and height

`timeLapse(slice_idx, dataset, timerange = None, plotfile='default')` -- A function for generating a video for a given time range

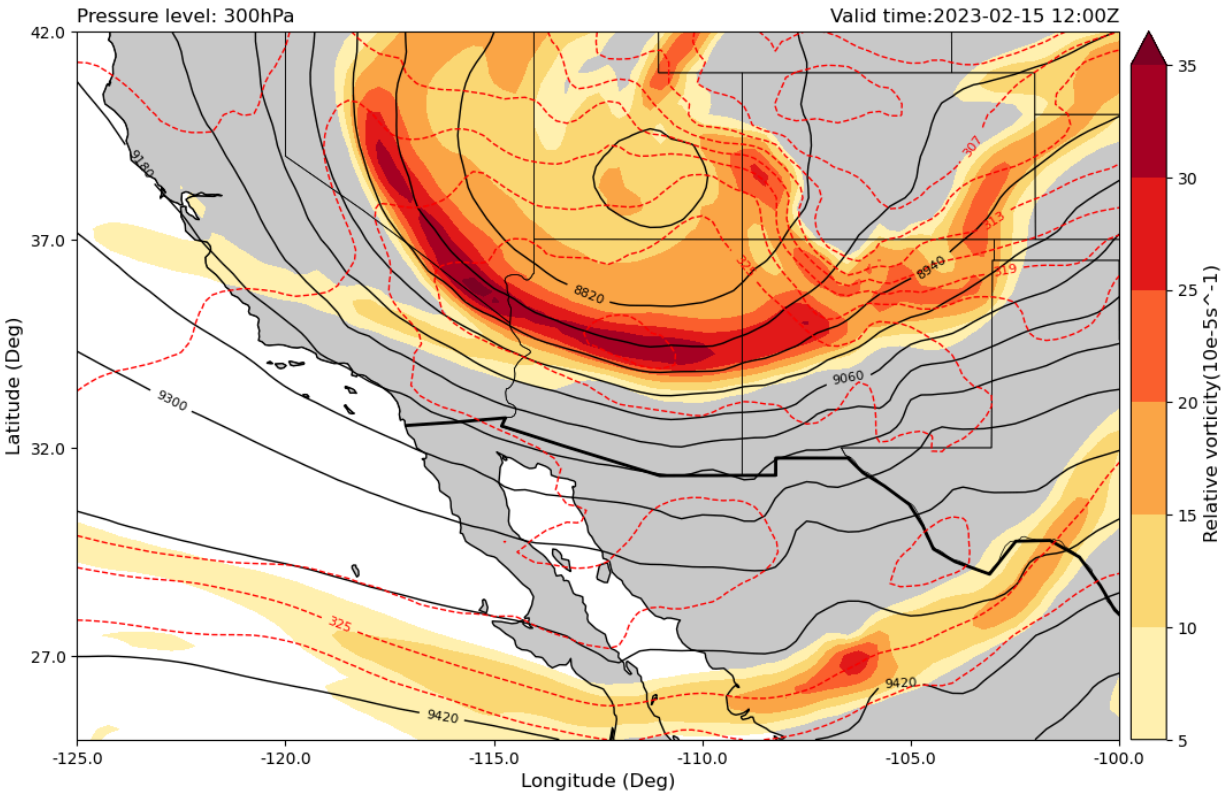
`scanner_p(slice_idx, dataset, coords = "all", TtoB = True, plotfile="default")` -- A function that can return a video that records plots at one area in different height

The following is the example for above functions.

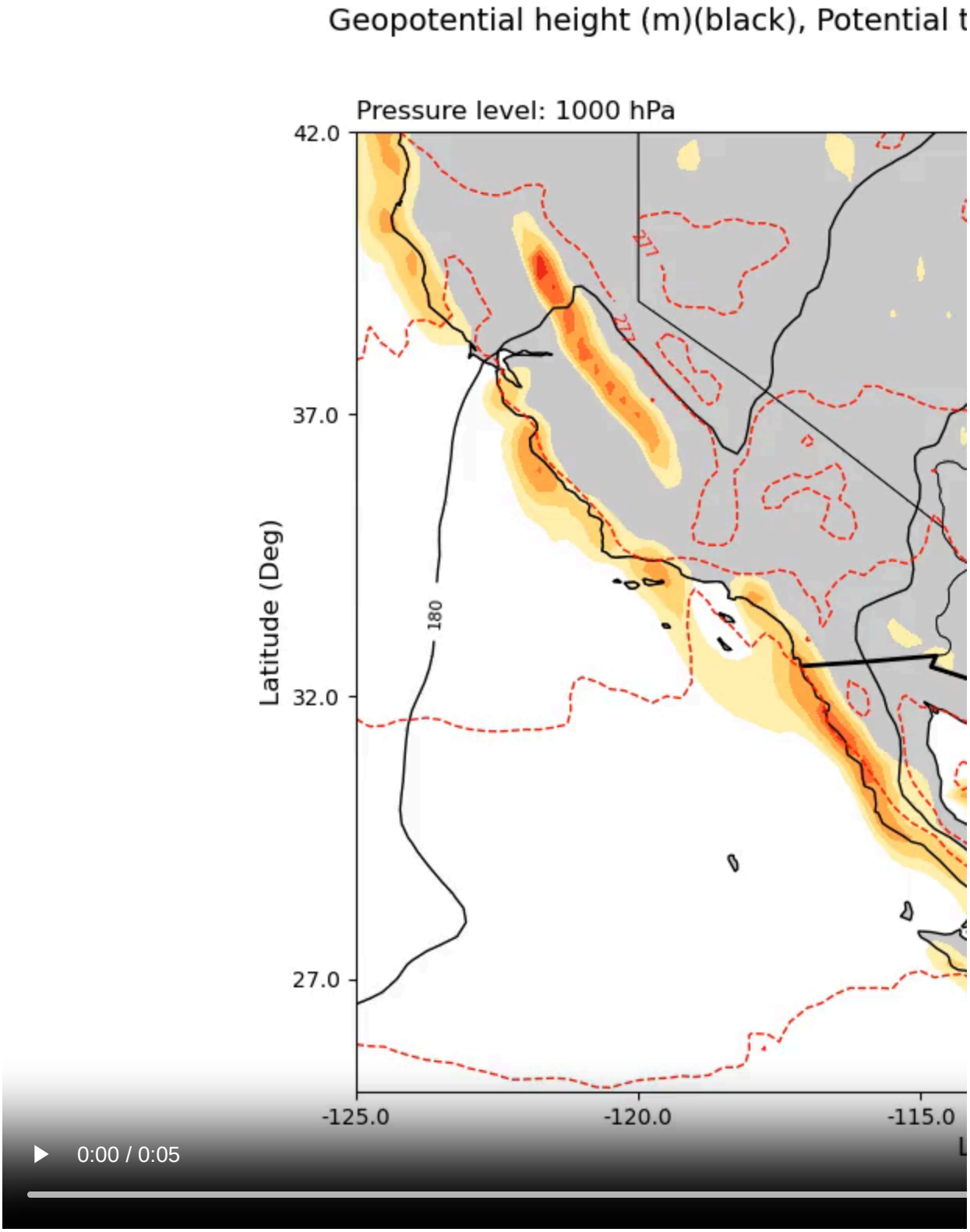
In [ ]:
```
ds1=Scanner.selection(ds, ['z', 'thta', 'vo'], extent=[-125,-100,25, 42])
Scanner.baseplot(None, ds1.sel(pressure = 300))
```

Out[ ]:
```
<GeoAxes: title={'left': 'Pressure level: 300hPa', 'right': 'Valid time:2023-02-15 12:00Z'}, xlabel='Longitude (Deg)', ylabel='Latitude (Deg)'>
```

Geopotential height (m)(black), Potential temperature (K)(red), Relative vorticity(10e-5s^-1)



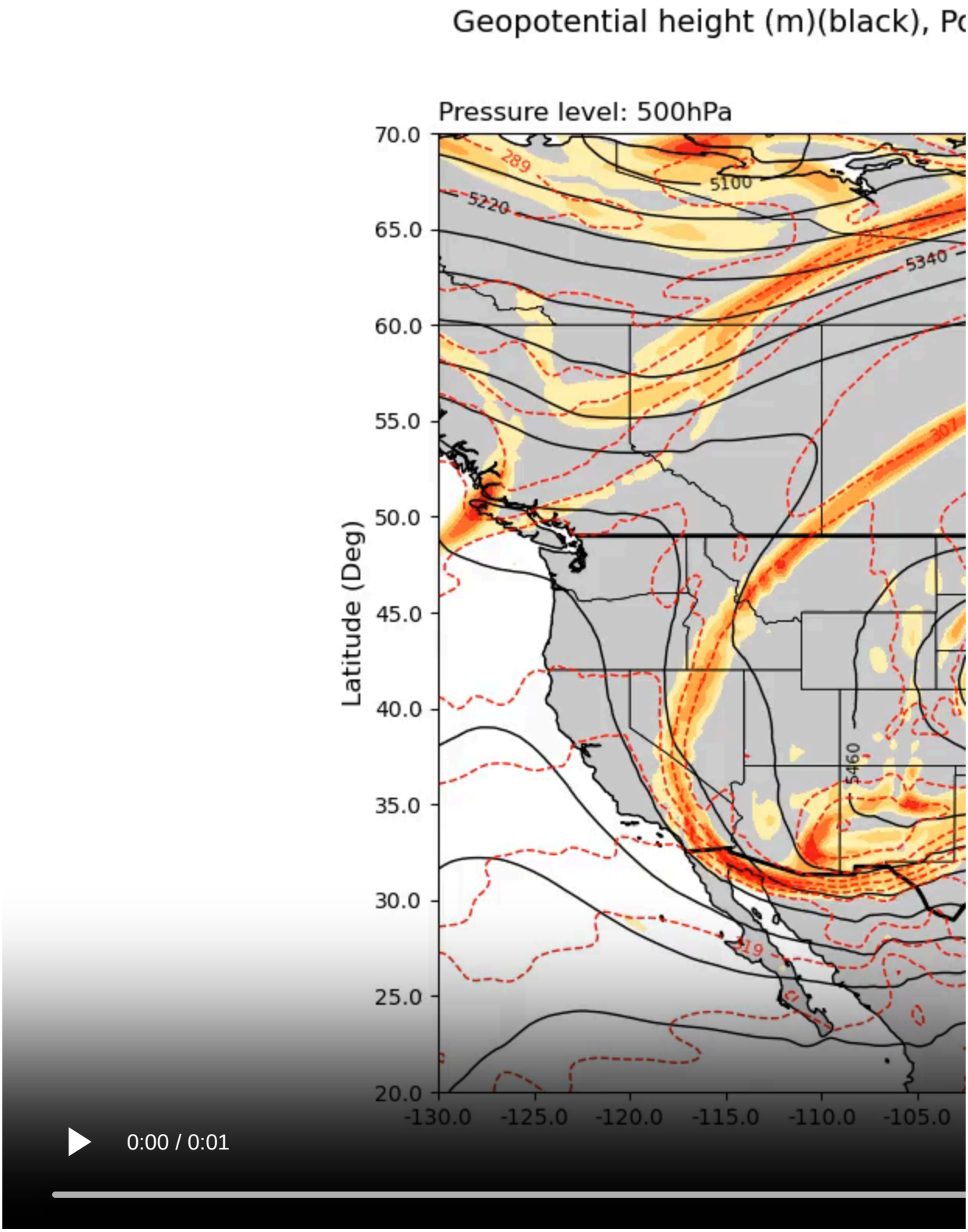
In [ ]:
```
Scanner.scanner_p(None, ds1)
```



<Figure size 640x480 with 0 Axes>

In [ ]:
```
ds=xr.open_dataset('./testdata/1214.grib', engine='cfgrib')
ds=ds.rename({"isobaricInhPa":"pressure","latitude":"lat", "longitude":"lon"})
ds['thta']=mpcalc.potential_temperature(ds.pressure, ds.t)
ds['z']=ds.z/9.8
ds2=Scanner.selection(ds, ['z', 'thta', 'vo'], plevel=500)
Scanner.timeLapse(None, ds2.isel(time = slice(1, 6)))
```

```
Ignoring index file './testdata/1214.grib.923a8.idx' incompatible with GRIB fi
le
```




```
<Figure size 640x480 with 0 Axes>
```

Last words

This notebook's objective is not only making cross section from example, but also creating standardized function for doing such work by any data from the ERA5 database. However, the original vision of this project is even further--creating standerized functions that can take any atmospheric or oceanic .grib format data to do cross section scanning. I still want this to be my half way goal--as all of these functions are planned to be in a module that can do atmospheric and oceanolographic scanning in different perspectives (Virtical Crossection, Horizontal Elevation, and Time lapse of a given layer and location), and eventually use for generating plots for my future research and my personal website.