MR2020: Coding for METOC

Module 13: Plotting with matplotlib

What is matplotlib?

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It largely emulates most of the functionality of MATLAB plotting.

Matplotlib provides control over every aspect of a figure, from plots and axes to titles and labels.

What are the main parts of a figure?

Figure

1. The entire window or page where a plot is drawn.
2. Can contain multiple Axes (subplots).

Axes:

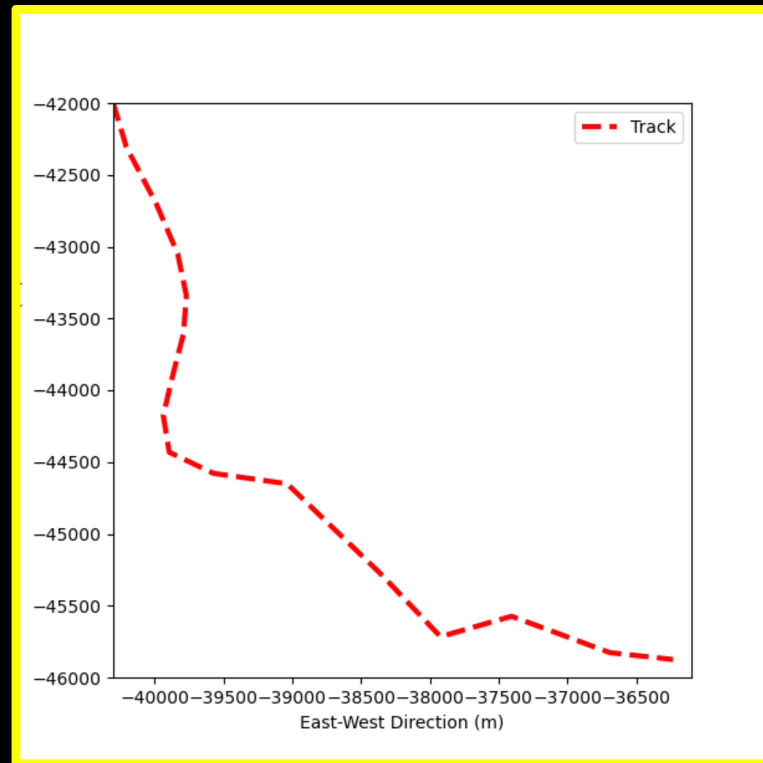
1. The area where data is plotted (essentially a subplot).
2. Contains X and Y axis, labels, titles, and more.

Plotting Functions:

1. High-level plotting functions like plot(), scatter(), bar(), and hist().

Customizations:

1. Control plot appearance with colors, markers, line styles, and more.



What are the main parts of a figure?

Figure

1. The entire window or page where a plot is drawn.
2. Can contain multiple Axes (subplots).

Axes

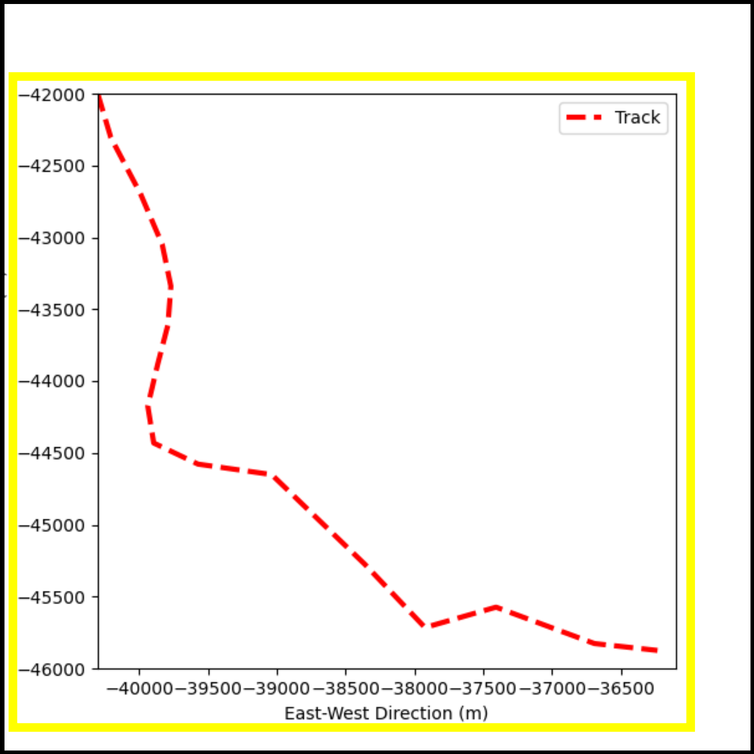
1. The area where data is plotted (essentially a subplot).
2. Contains X and Y axis, labels, titles, and more.

Plotting Functions:

1. High-level plotting functions like plot(), scatter(), bar(), and hist().

Customizations:

1. Control plot appearance with colors, markers, line styles, and more.



What are the main parts of a figure?

Figure

1. The entire window or page where a plot is drawn.
2. Can contain multiple Axes (subplots).

Axes

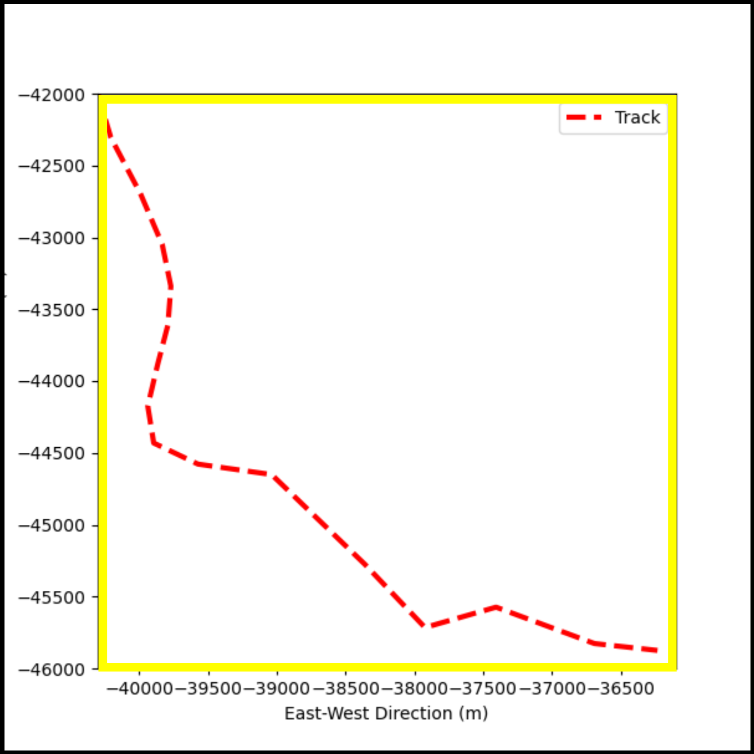
1. The area where data is plotted (essentially a subplot).
2. Contains X and Y axis, labels, titles, and more.

Plotting Functions

High-level plotting functions like plot(), scatter(), bar(), and hist().

Customizations

Control plot appearance with colors, markers, line styles, and more.



Making a Figure

We can also do `import matplotlib.pyplot as plt`
`cm` gives us some colormap options for shaded plots.

```
from matplotlib import pyplot as plt, cm

# The most basic command for making a line plot
# First input is the x-axis; second is the y-axis
plt.plot(x,y)
```

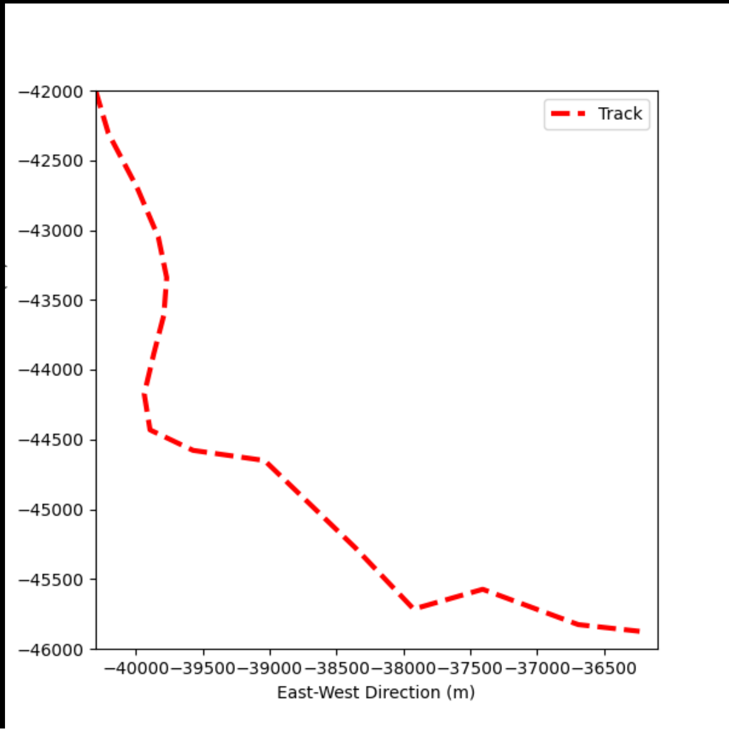
Doing plt.command does not give us much control over the details of the plot, however. For that we want control over both the figure and its axes.

```
fig, ax = plt.subplots(1,1,figsize=(6,6))
```

The subplots command will create one or more axes on a grid within a figure. Above, the command produces a figure with a single axis (1 row and 1 column). The `figsize` argument controls the size of the figure when it is saved/printed. In the example above, the figure is 6 by 6 inches. `ax` is now an object we can use to have fine-grained control over every aspect of the plot.

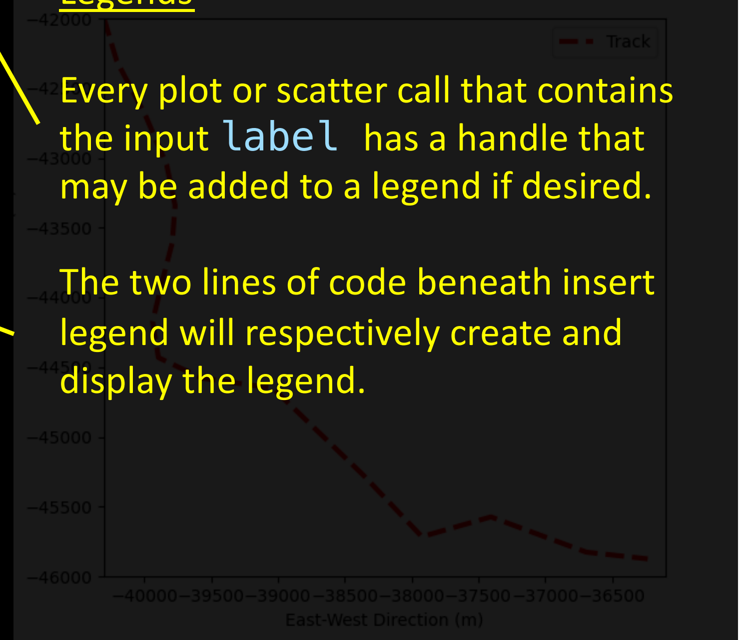
Line Plot (`plot`)

```
# Create the figure and axis object
fig, ax = plt.subplots(1,1,figsize=(6,6))

# Create line plot
# All parameters are optional.
# color can be chosen from many named colors
(https://matplotlib.org/stable/gallery/color/named_colors.html)
# ls is linestyle (- = solid; -- = dashed; : = dotted; -. = dash-dotted)
# lw is linewidth
# label is the label for the line that will show up in legend
ax.plot(ex.x,ex.y,color='r',ls='--',lw=3,label='Track')

# Label x-axis
ax.set_xlabel('East-West Direction (m)')

# Label y-axis
ax.set_ylabel('North-South Direction (m)')

# Insert legend
h, l = ax.get_legend_handles_labels()
ax.legend()

# Set limits to x and y-axes
ax.set_xlim([-40300,-36100])
ax.set_ylim([-46000,-42000])

# Save figure
fig.savefig('lineplot.png')
```



Displaying Legends

```
# Create the figure and axis object
fig, ax = plt.subplots(1,1,figsize=(6,6))

# Create line plot
# All parameters are optional.
# color can be chosen from many named colors
(https://matplotlib.org/stable/gallery/color/named_colors.html)
# ls is linestyle (- = solid; -- = dashed; : = dotted; -. = dash-dotted)
# lw is linewidth
# label is the label for the line that will show up in legend
ax.plot(ex.x,ex.y,color='r',ls='--',lw=3,label='Track')

# Label x-axis
ax.set_xlabel('East-West Direction (m)')

# Label y-axis
ax.set_ylabel('North-South Direction (m)')

# Insert legend
h, l = ax.get_legend_handles_labels()
ax.legend()

# Set limits to x and y-axes
ax.set_xlim([-40300,-36100])
ax.set_ylim([-46000,-42000])

# Save figure
fig.savefig('lineplot.png')
```



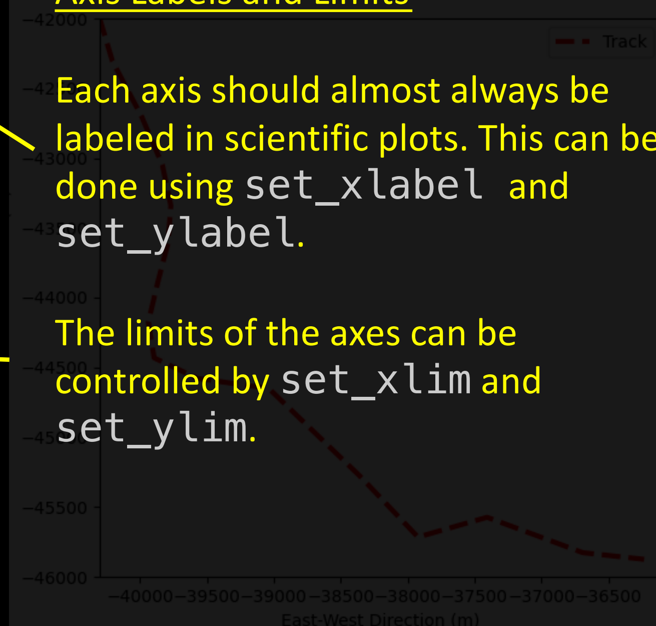
Legends

Every plot or scatter call that contains the input `label` has a handle that may be added to a legend if desired.

The two lines of code beneath insert legend will respectively create and display the legend.

Manipulating Axis Labels and Limits

```
# Create the figure and axis object
fig, ax = plt.subplots(1,1,figsize=(6,6))

# Create line plot
# All parameters are optional.
# color can be chosen from many named colors
(https://matplotlib.org/stable/gallery/color/named_colors.html)
# ls is linestyle (- = solid; -- = dashed; : = dotted; -. = dash-dotted)
# lw is linewidth
# label is the label for the line that will show up in legend
ax.plot(ex.x,ex.y,color='r',ls='--',lw=3,label='Track')

# Label x-axis
ax.set_xlabel('East-West Direction (m)')

# Label y-axis
ax.set_ylabel('North-South Direction (m)')

# Insert legend
h, l = ax.get_legend_handles_labels()
ax.legend()

# Set limits to x and y-axes
ax.set_xlim([-40300,-36100])
ax.set_ylim([-46000,-42000])

# Save figure
fig.savefig('lineplot.png')
```



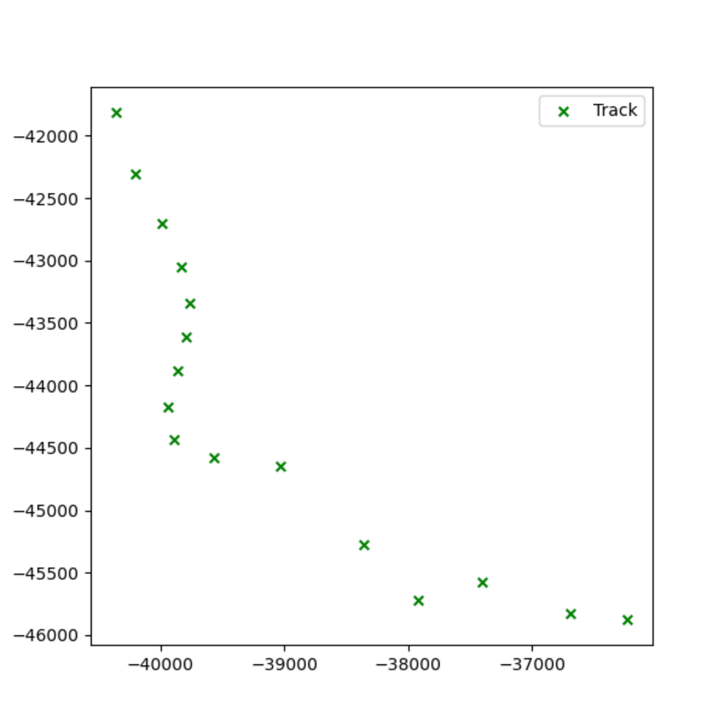
Axis Labels and Limits

Each axis should almost always be labeled in scientific plots. This can be done using `set_xlabel` and `set_ylabel`.

The limits of the axes can be controlled by `set_xlim` and `set_ylim`.

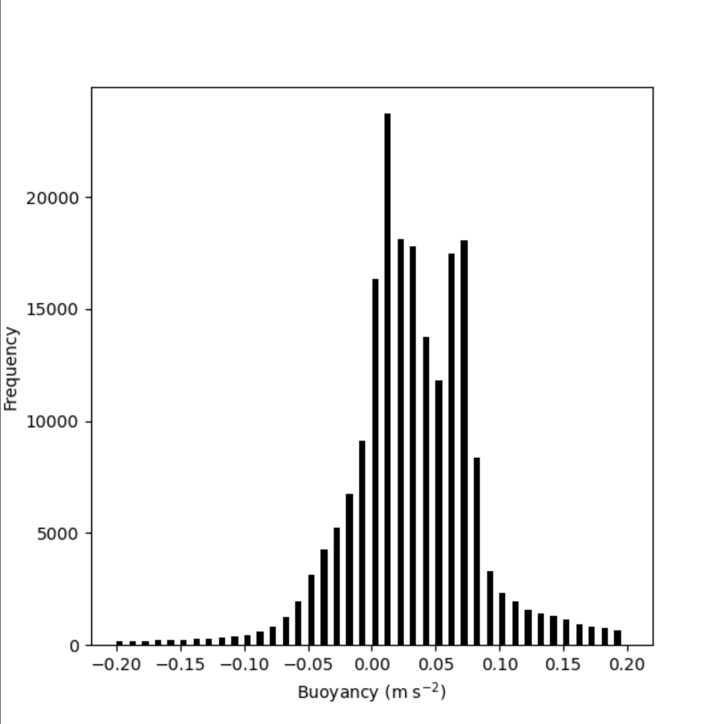
Scatter Plot (`scatter`)

```python
# Making a scatter plot

fig, ax = plt.subplots(1,1,figsize=(6,6))

# marker: What symbol is used to denote point
# s: size of marker
ax.scatter(ex.x,ex.y,color='g',marker='x',s=30,label='Track')

ax.set_xlabel('East-West Direction (m)')
ax.set_ylabel('North-South Direction (m)')

h, l = ax.get_legend_handles_labels()
ax.legend()

fig.savefig('scatter.png')
```



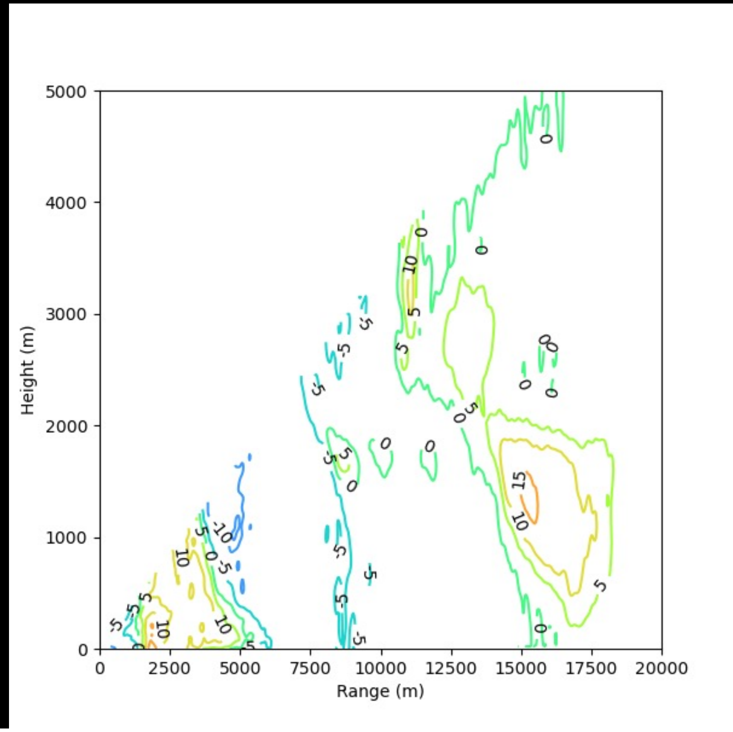
Bar Plot/Histogram (`hist`)

```
import numpy as np

# Create bins for histogram. The values represent the left end of the bin.
# The step size (0.01) denotes width of each bar.
bins = np.arange(-0.2,0.21,0.01)

fig, ax = plt.subplots(1,1,figsize=(6,6))

# By default, the width will be the width of the bin.
# But width can be manually reset as seen below.
ax.hist(df.b,bins=bins,color='k',width=0.005)

ax.set_xlabel(r'Buoyancy (m s$^{-2}$)')
ax.set_ylabel('Frequency')

fig.savefig('histogram.png')
```



Contour Plot (`contour`)

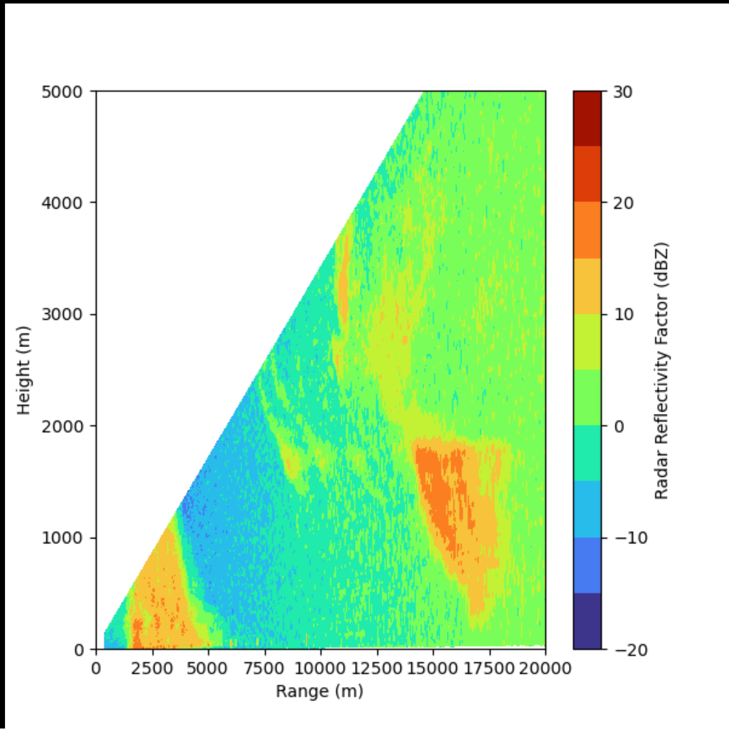
```
fig, ax = plt.subplots(1,1,figsize=(6,6))

c = ax.contour(r,H,dbzsmooth,np.arange(-20,35,5),cmap=cm.get_cmap('turbo'))
ax.set_xlim([0,20000])
ax.set_ylim([0,5000])
ax.set_xlabel('Range (m)')
ax.set_ylabel('Height (m)')
ax.clabel(c, inline=True, colors='k',fontsize=10, fmt="%d")
# fmt specifies the format of the label

fig.savefig('contour.png')
```



Filled Contour Plot (`contourf`)

```
fig, ax = plt.subplots(1,1,figsize=(6,6))

c = ax.contourf(r,H,dbz,np.arange(-20,35,5),cmap=cm.get_cmap('turbo'))
ax.set_xlim([0,20000])
ax.set_ylim([0,5000])
ax.set_xlabel('Range (m)')
ax.set_ylabel('Height (m)')

cb = fig.colorbar(c,ax=ax,label='Radar Reflectivity Factor (dBZ)')

fig.savefig('contourf.png')
```

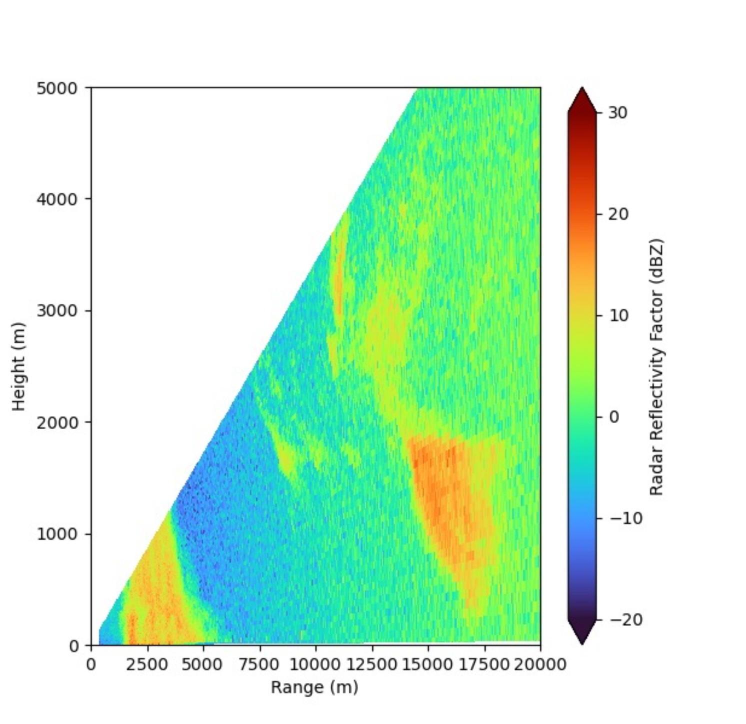
NOTE: For most 2D data, you will want to use a color mesh plot (next slides). The contour plot may appear smoother because it interpolates to draw the contour lines. This can cause the contourf plots to misrepresent data.



Color Mesh Plot (pcolormesh)

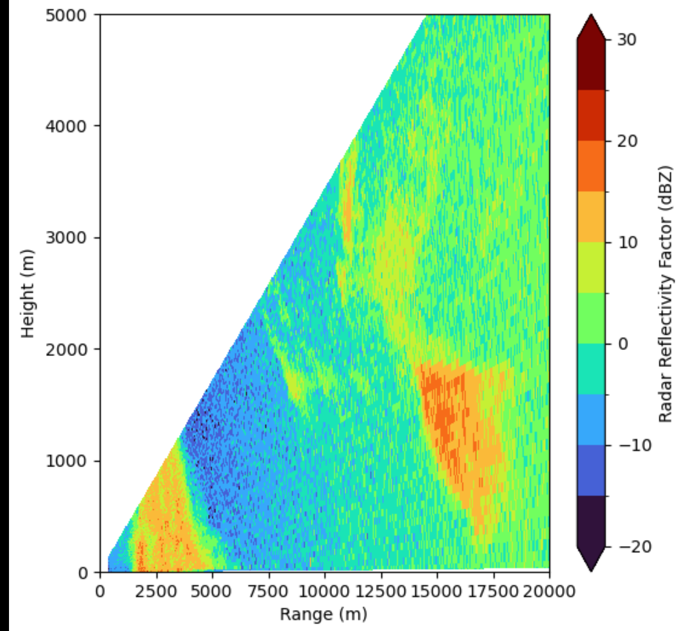
```
fig, ax = plt.subplots(1,1,figsize=(6,6))

c = ax.pcolormesh(r,H,dbz,vmin=-20,vmax=30,cmap=cm.get_cmap('turbo'))
ax.set_xlim([0,20000])
ax.set_ylim([0,5000])
ax.set_xlabel('Range (m)')
ax.set_ylabel('Height (m)')

cb = fig.colorbar(c,ax=ax,extend='both',label='Radar Reflectivity Factor (dBZ)')

fig.savefig('colormesh_continuous.png')
```

NOTE: A color mesh plot is preferred for 2D shaded data because each data point is individually represented. No interpolation like contourf.



Using a Discrete Colorbar with 2D Mesh Plots

```
# Define the discrete boundaries for color bins
boundaries = np.arange(-20, 35, 5) # Bins from -20 to 30 with step size 5
norm = colors.BoundaryNorm(boundaries, ncolors=cm.get_cmap('turbo').N)

fig, ax = plt.subplots(1,1,figsize=(6,6))

c = ax.pcolormesh(r,H,dbz,cmap=cm.get_cmap('turbo'),norm=norm)
ax.set_xlim([0,20000])
ax.set_ylim([0,5000])
ax.set_xlabel('Range (m)')
ax.set_ylabel('Height (m)')
cb = fig.colorbar(c,ax=ax,extend='both',label='Radar Reflectivity Factor (dBZ)')


fig.savefig('colormesh_discrete.png')
```



Violin Plot (`violinplot`)

```
# Generate example data
data = [
np.random.exponential(4, 50), # Category A
np.random.normal(1, 1.5, 100), # Category B
np.random.gamma(5,1,100) # Category C
]

# Create the violin plot
fig, ax = plt.subplots(figsize=(6, 6))
ax.violinplot(data, showmeans=True, showmedians=True)

# Customize the plot
ax.set_title('Violin Plot Examples using Matplotlib')
ax.set_xlabel('Distribution')
ax.set_ylabel('Value')
ax.set_xticks([1, 2, 3])
ax.set_xticklabels(['Exponential', 'Normal', 'Gamma'])

fig.savefig('violinplot.png')
```