SECTION 3: TWO-DIMENSIONAL PLOTTING

ENGR 103 – Introduction to Engineering Computing

Data Visualization

- Like it or not, the ability to ***communicate effectively*** is an important aspect of being a successful engineer
  - Coworkers, managers, marketing, customers, etc.

- As engineers, effective communication often means ***effective communication of data***
  - Technical writing
  - ***Graphical presentation of data***: plots, graphs, charts, etc.

- Python has a variety of data-visualization tools available
  - We will use the ***Pyplot*** module from within the ***Matplotlib*** package

- Plots fall into two main categories:
  - **2-D plotting** – we'll introduce these plots here
  - **3-D plotting** – covered later in the course

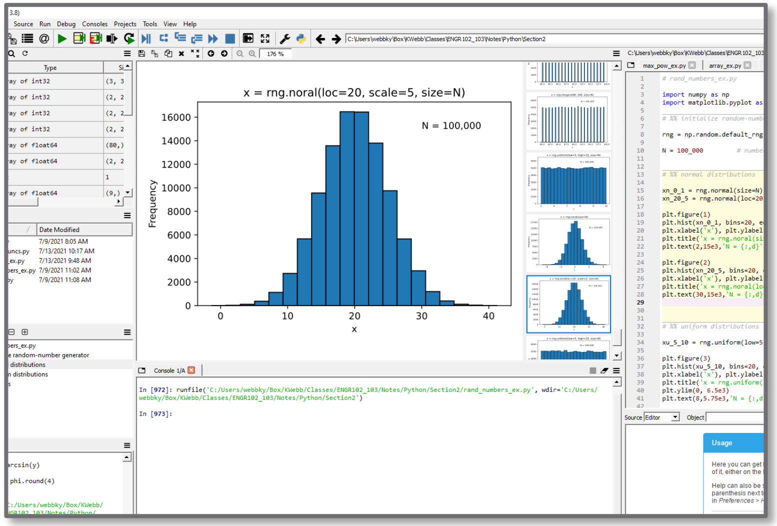
Matplotlib and Pyplot

- ***Matplotlib***
  - Python package or library for creating a wide variety of plots
- ***Pyplot***
  - Matplotlib module including all of the plot functions we will use (and many, many more)
- As usual, we must ***import*** Matplotlib and Pyplot before we can use them
  - Import only the Pyplot module in any script where plots will be created:

```
from matplotlib import pyplot as plt
```

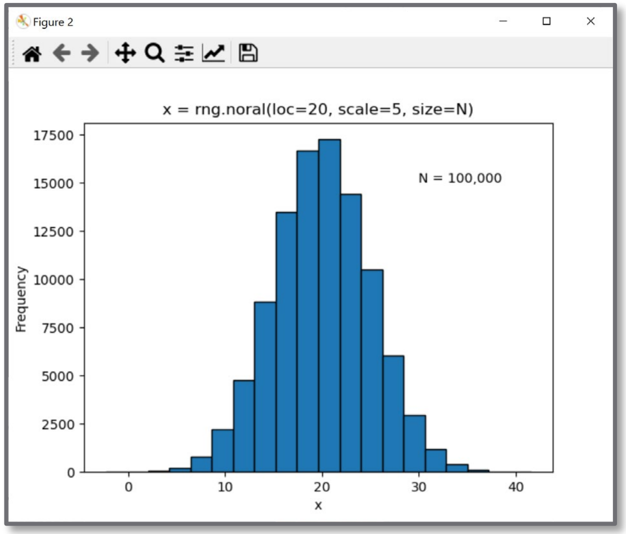
Plotting In Spyder

- Two primary modes of displaying plots within Spyder:
  - ***Inline***
    - Plot pane docked in the Spyder interface
    - Not interactive (cannot pan, zoom, measure, etc.)
  - ***Automatic***
    - Plots created in a separate window
    - Interactive (can pan, zoom, measure, etc.)



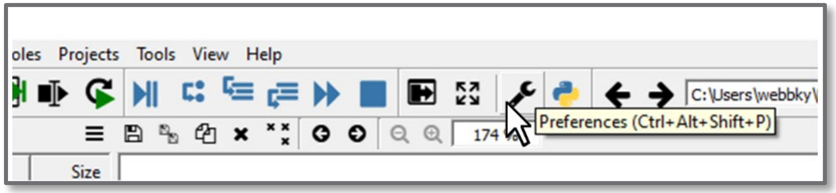
```
# rand_numbers_ex.py

import numpy as np
import matplotlib.pyplot as

# %% initialize random-numb

rng = np.random.default_rng

N = 100_000         # numbe


# %% normal distributions

xn_0_1 = rng.normal(size=N)
xn_20_5 = rng.normal(loc=20

plt.figure(1)
plt.hist(xn_0_1, bins=20, e
plt.xlabel('x'), plt.ylabel
plt.title('x = rng.noral(si
plt.text(2,15e3,'N = {:,d}'

plt.figure(2)
plt.hist(xn_20_5, bins=20,
plt.xlabel('x'), plt.ylabel
plt.title('x = rng.noral(lo
plt.text(30,15e3,'N = {:,d}



# %% uniform distributions

xu_5_10 = rng.uniform(low=5

plt.figure(3)
plt.hist(xu_5_10, bins=20,
plt.xlabel('x'), plt.ylabel
plt.title('x = rng.uniform(
plt.ylim(0, 6.5e3)
plt.text(8,5.75e3,'N = {:,d
```

```
In [972]: runfile('C:/Users/webbky/Box/KWebb/Classes/ENGR102_103/Notes/Python/Section2/rand_numbers_ex.py', wdir='C:/Users/
webbky/Box/KWebb/Classes/ENGR102_103/Notes/Python/Section2')

In [973]:
```

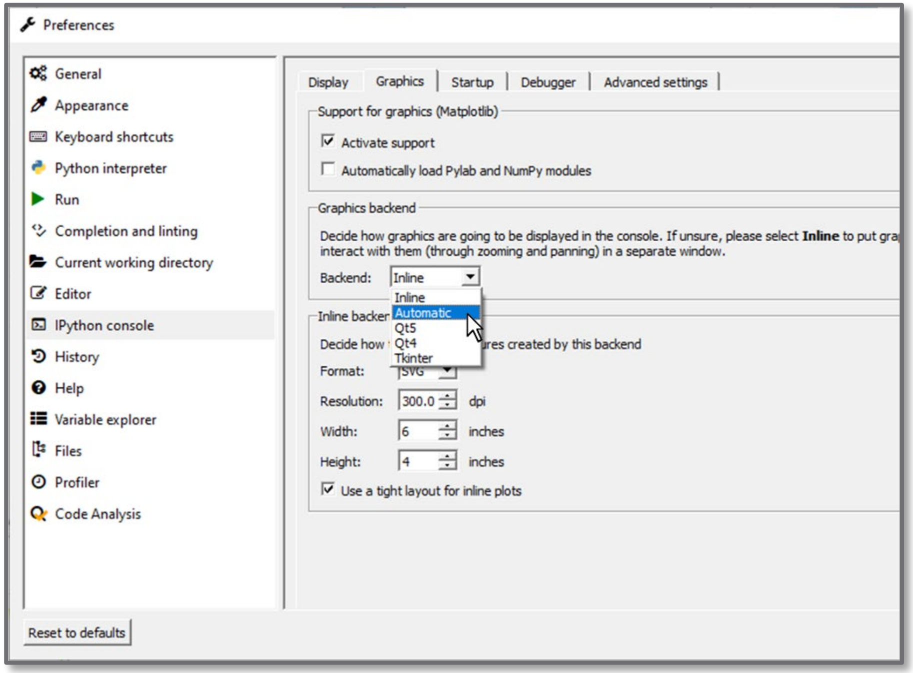


Plotting In Spyder

- To switch between Inline and Automatic plotting:
  - Preferences



  - IPython Console:
    - Graphics
    - Backend
      - Inline
      - Automatic



Support for graphics (Matplotlib)

☑ Activate support

☐ Automatically load Pylab and NumPy modules

Graphics backend

Decide how graphics are going to be displayed in the console. If unsure, please select **Inline** to put gra
interact with them (through zooming and panning) in a separate window.

Backend: Inline

Inline
Automatic
Qt5
Qt4
Tkinter

Inline backer

Decide how ures created by this backend

Format: SVG

Resolution: 300.0 dpi

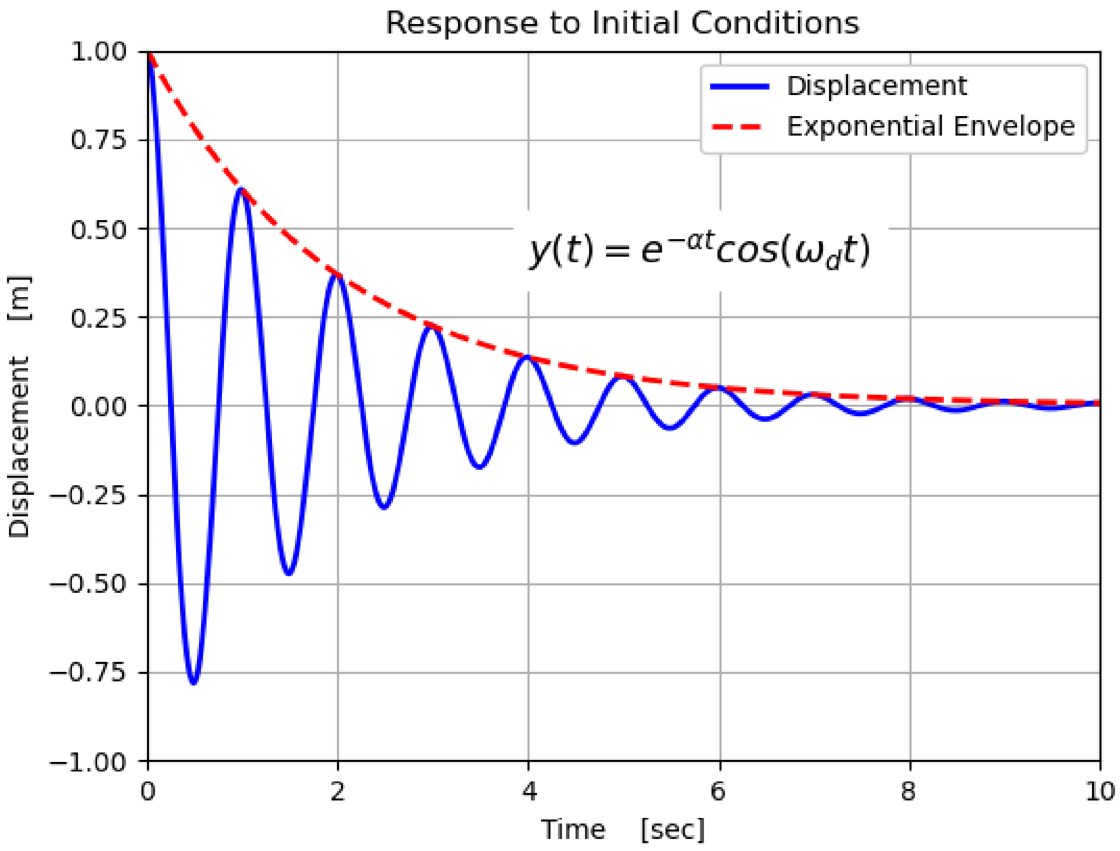
Width: 6 inches

Height: 4 inches

☑ Use a tight layout for inline plots

2-D Line Plots

Basic 2-D Plotting – `plt.plot()`



Basic 2-D Plotting – `plt.plot()`

- Syntax:

```
plt.plot(x, y, fmt, **kwargs)
```

  - `x`: *optional* – abscissa – horizontal-axis data
  - `y`: ordinate – vertical-axis data
    - x and y are equal-length vectors
  - `fmt`: *optional* – format specification string – defines:
    - Line type – e.g. solid, dashed, dotted
    - Line color
    - Marker shape – placed at each data point
  - `**kwargs:` *optional* - arbitrary number of keyword/argument pairs, e.g.
    - linewidth=2
    - markersize=8

plot() – fmt – Line Style

```
plt.plot(x, y, fmt, **kwargs)
```

- Three components – ***line style***, ***marker***, ***color***
  - Each is *optional* – specify some or all
- ***Line Style*** specifiers:

Specifier	Line Style
'-'	Solid
'--'	Dashed
':'	Dotted
'-.'	Dash-dot

- Default is a solid line

`plot()` – `fmt` – Marker

- ***Marker*** **specifiers:**
  - A partial list

Specifier	Marker
`'+'`	Plus sign
`'o'`	Circle
`'*'`	Asterisk
`'.'`	Point
`'x'`	Cross
`'s'`	Square
`'d'`	Diamond

Specifier	Marker
`'^'`	Upward-pointing triangle
`'v'`	Downward-pointing triangle
`'>'`	Right-pointing triangle
`'<'`	Left-pointing triangle
`'p'`	pentagon
`'h'`	hexagon

- Default is no marker
  - Markers are placed at every data point – can get crowded for closely spaced data

plot() – fmt – Line/Marker Color

- ***Color*** specifiers:

Specifier	Color
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan

Specifier	Color
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

- Default color is blue
  - If multiple x,y pairs are specified in a single plot command, line/marker colors will cycle through automatically (white is skipped for white background)

plot() - **kwargs

```
plt.plot(x, y, fmt, **kwargs)
```

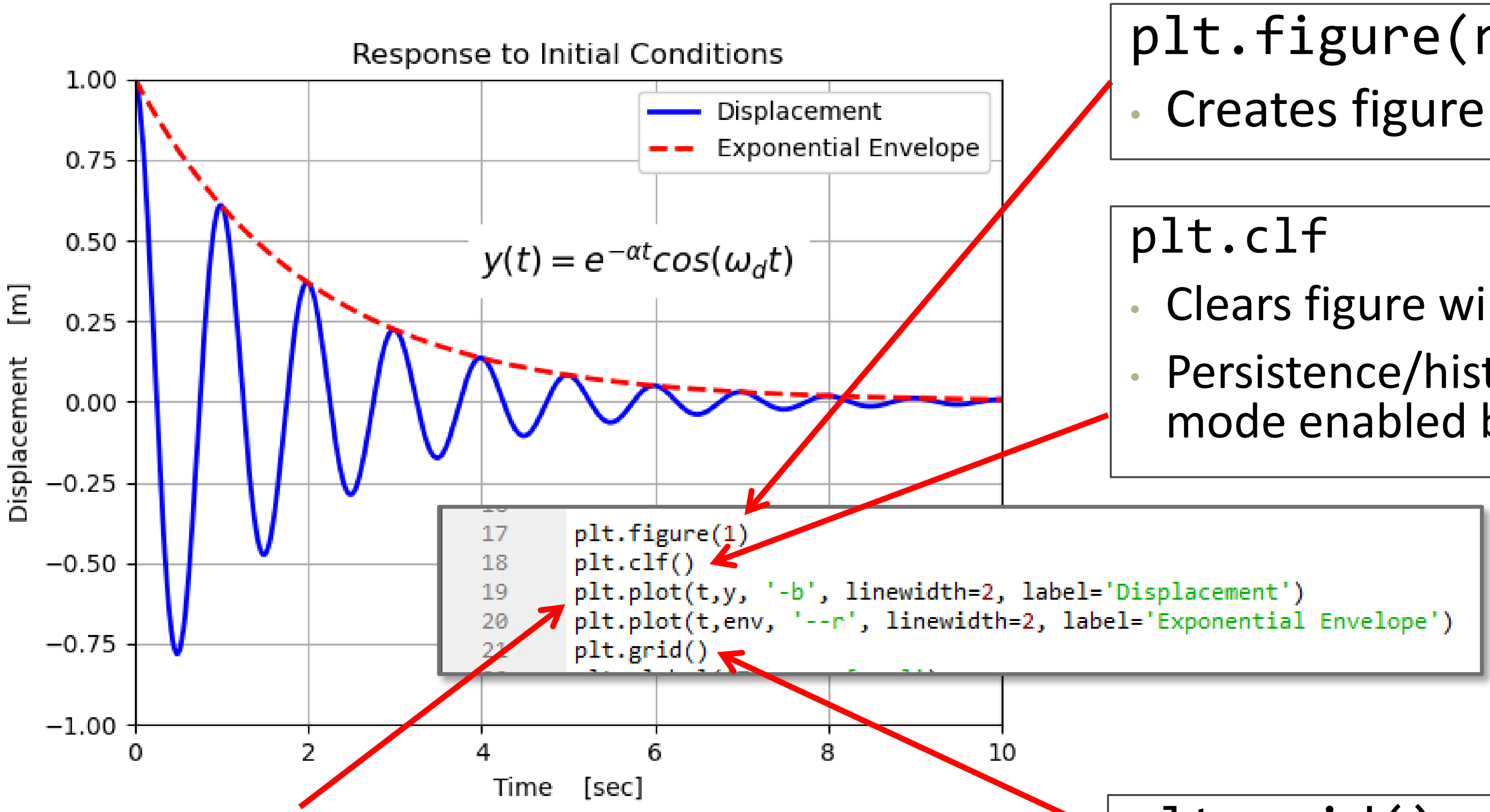
- ***Keyword/value pairs*** – a few examples:
  - `linewidth` (`lw`): width of line – points
    - `linewidth=2`
  - `markeredgecolor` (`mec`): color of the marker or edge color for filled markers - a string
    - `markeredgecolor='k'`
  - `markerfacecolor` (`mfc`): face color of filled markers – a string
    - `markerfacecolor='g'`

plot() - **kwargs

```
plt.plot(x, y, fmt, **kwargs)
```

- ***Keyword/value pairs*** – (continued):
  - markersize (ms): size of markers – points
    - markersize=8
  - label: string attached to the plot that will be displayed in the legend
    - label='displacement'

Using plot()



```
plt.figure(1)
plt.clf()
plt.plot(t,y, '-b', linewidth=2, label='Displacement')
plt.plot(t,env, '--r', linewidth=2, label='Exponential Envelope')
plt.grid()
```

`plt.figure(n)`

- Creates figure window

`plt.clf`

- Clears figure window
- Persistence/history/hold mode enabled by default

`plt.plot()`

- Plots the data

`plt.grid()`

- Turns on grid lines

Plot Annotation

Plot Annotation

- ***Title***

```
plt.title(‘string’, **kwargs)
```

- ***Axis labels***

```
plt.xlabel(‘string’, **kwargs)
plt.ylabel(‘string’, **kwargs)
```

- ***Text***

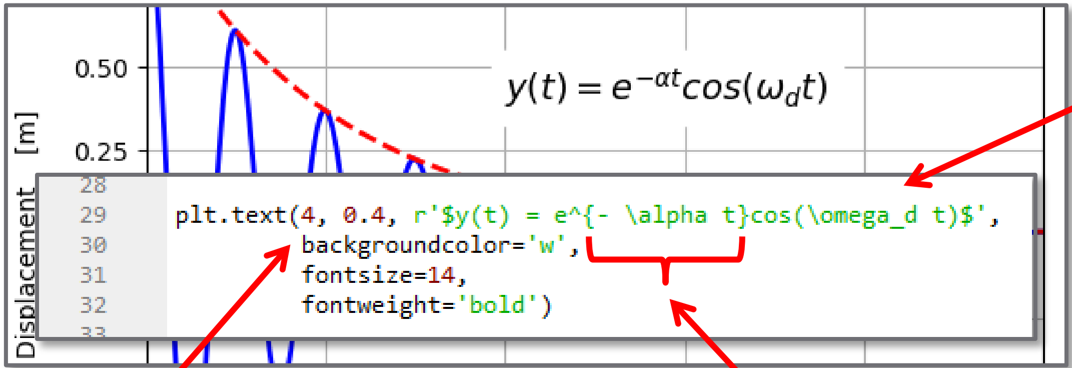
```
plt.text(x,y,‘string’, **kwargs)
```

  - Text string printed at location (x,y) on the current figure axes

- *****kwargs***
  - Keyword/value pairs specifying text properties
  - Common to all annotation functions
  - For example: `color='r'`, `backgroundcolor='w'`, `fontsize=12`, etc.

Plot Annotation – Mathematical Expressions

- Matplotlib can interpret TeX character sequences
  - E.g. `\beta`, `\mu`, `\omega`, `\div`, etc.
  - Search Matplotlib documentation for ‘mathematical expressions’ for more information
  - Precede opening string quote with an `r` to create a ***raw string***
  - Enclose mathematical expressions in dollar signs, $ … $



```
plt.text(4, 0.4, r'$y(t) = e^{- \alpha t}cos(\omega_d t)$',
           backgroundcolor='w',
           fontsize=14,
           fontweight='bold')
```

Subscript
- Underscore, _
- Enclose multiple characters in curly brackets, { … }

Superscript
- Carrot, ^
- Enclose multiple characters in curly brackets, { … }

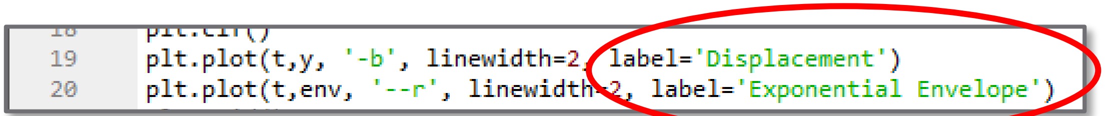
Backgroundcolor='…'
- Improves readability with gridlines

Plot Legend

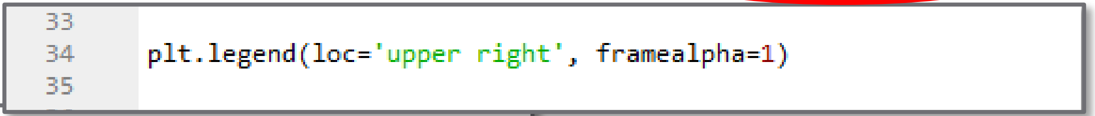
- Add a legend to a figure to identify multiple traces

```
plt.legend(**kwargs)
```

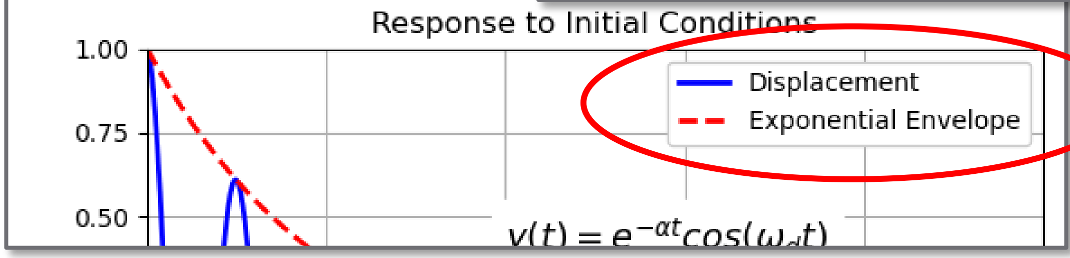
- `**kwargs`: optional keyword/value arguments, e.g.:
  - `loc='best'`, `title='Damping Ratio'`, `fontsize=12`
- Legend labels defined by `label='…'` in `plt.plot()` commands



```
plt.clf()
plt.plot(t,y, '-b', linewidth=2, label='Displacement')
plt.plot(t,env, '--r', linewidth=2, label='Exponential Envelope')
```



```

plt.legend(loc='upper right', framealpha=1)

```



plt.legend() - **kwargs

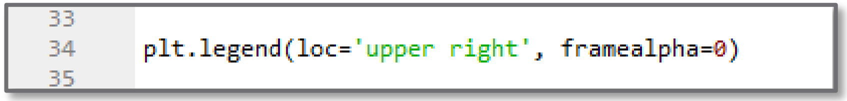
- A few useful keyword arguments:
- `loc='location string'`
  - Placement of legend box within the plot axes
  - Default: `loc='best'` – (placed to minimize overlap)
  - Can alternatively specify numeric location code
  - 'right' and 'center right' are the same

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5

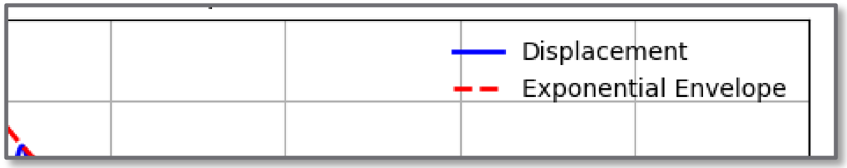
Location String	Location Code
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

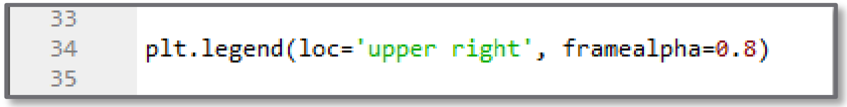
plt.legend() - **kwargs

- framealpha=α
  - Opacity of legend box background and frame
    - α=0 - completely transparent background, no frame
    - α=1 - completely opaque background, frame
  - Useful for blocking gridlines to improve readability
  - Default: framealpha=0.8

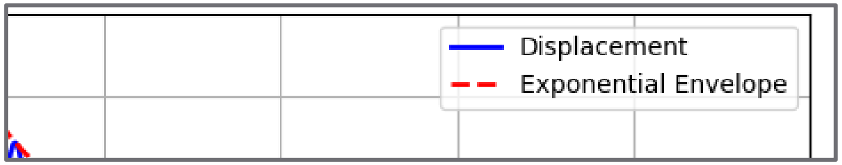


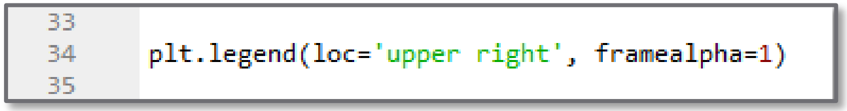
```
plt.legend(loc='upper right', framealpha=0)
```



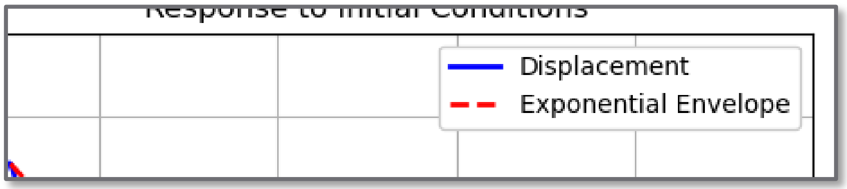


```
plt.legend(loc='upper right', framealpha=0.8)
```





```
plt.legend(loc='upper right', framealpha=1)
```



2-D Line Plots

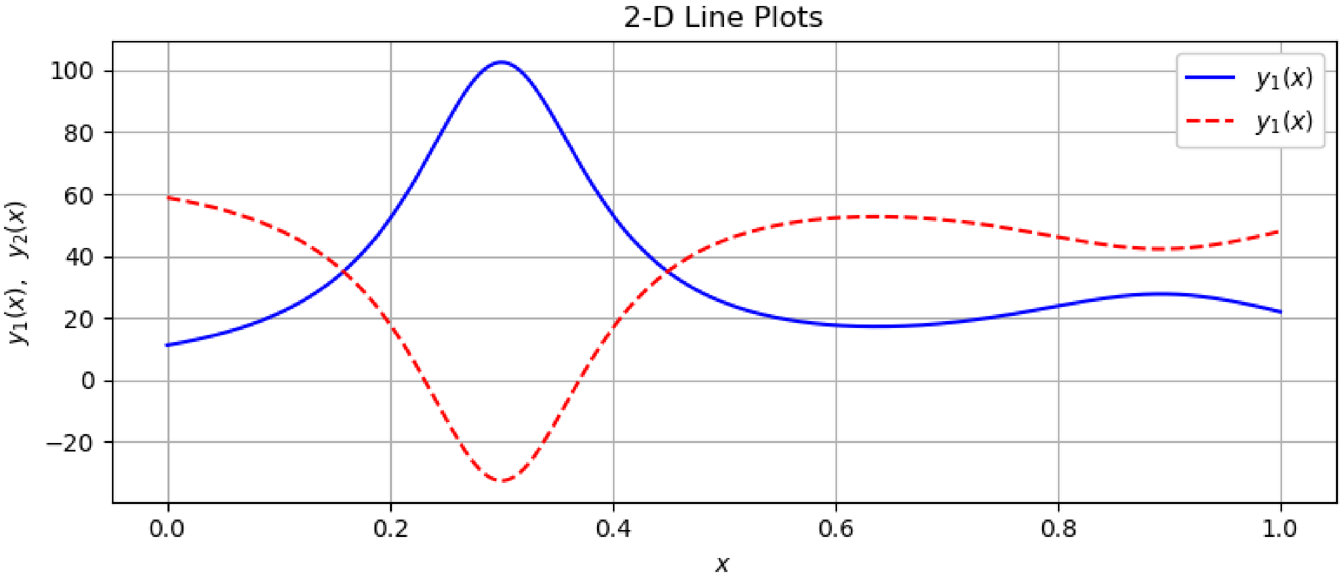
Exercise

- Write a script to do the following:
  - Create and x vector from 0 to 1 with 2000 points
  - Create vectors y_1 and y_2:

$$y_1 = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.04}$$

$$y_2 = -y_1 + 70$$

  - Create the following plot:



Subplots

Subplots

- Plot ***multiple sets of axes*** on a single figure

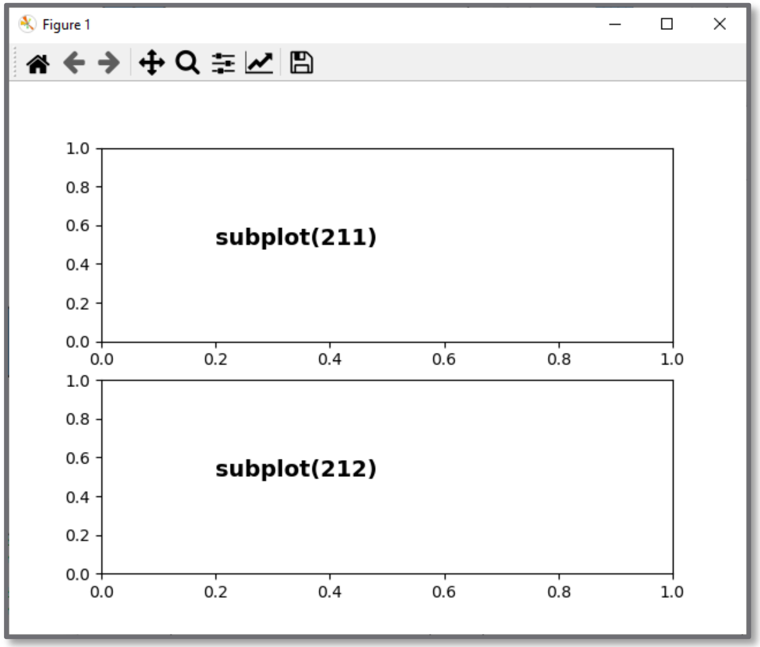
```
ax = plt.subplot(m,n,p)
ax = plt.subplot(mnp)
```

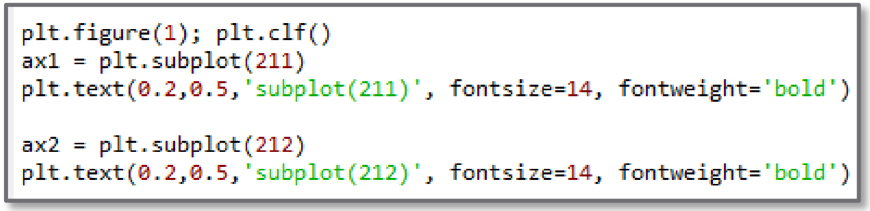
  - `m`: number of rows of axes in the figure window
  - `n`: number of columns of axes in the figure window
  - `p`: index of the currently active subplot – counted left-right, top-bottom
  - `ax`: *optional* – axis object pointing to the indexed axes

- `subplot` command activates the p^th^ subplot
  - All subsequent plotting/annotation commands issued to the active subplot
  - To plot to another subplot, call `subplot` again, with a new value for p

Subplot Numbering

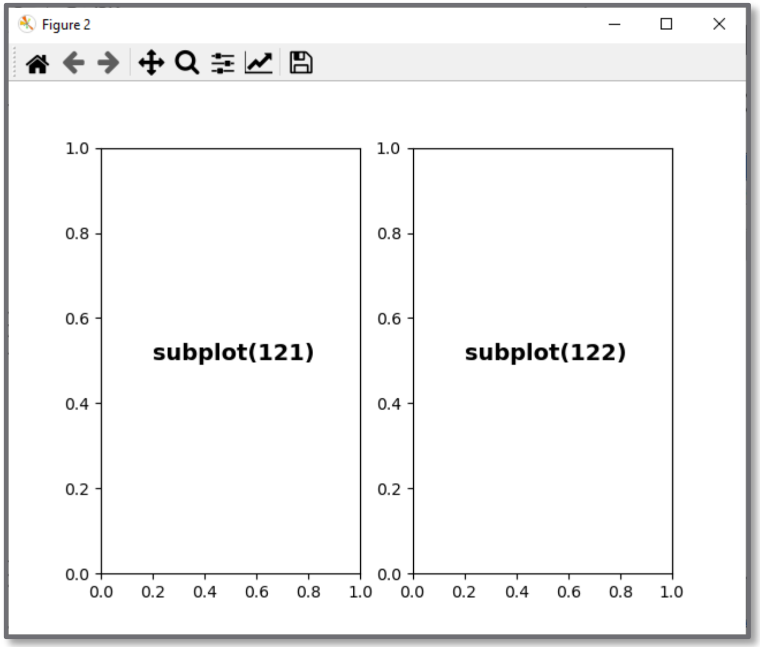
- □ 2 rows, 1 column

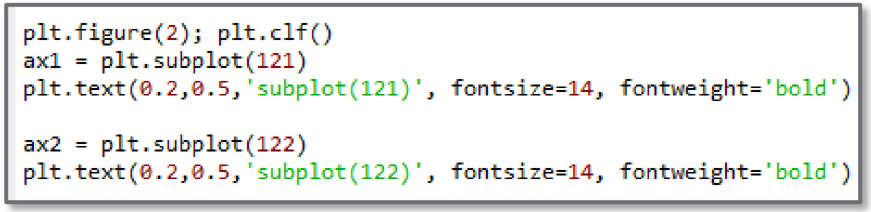




```
plt.figure(1); plt.clf()
ax1 = plt.subplot(211)
plt.text(0.2,0.5,'subplot(211)', fontsize=14, fontweight='bold')

ax2 = plt.subplot(212)
plt.text(0.2,0.5,'subplot(212)', fontsize=14, fontweight='bold')
```

- □ 1 row, 2 columns

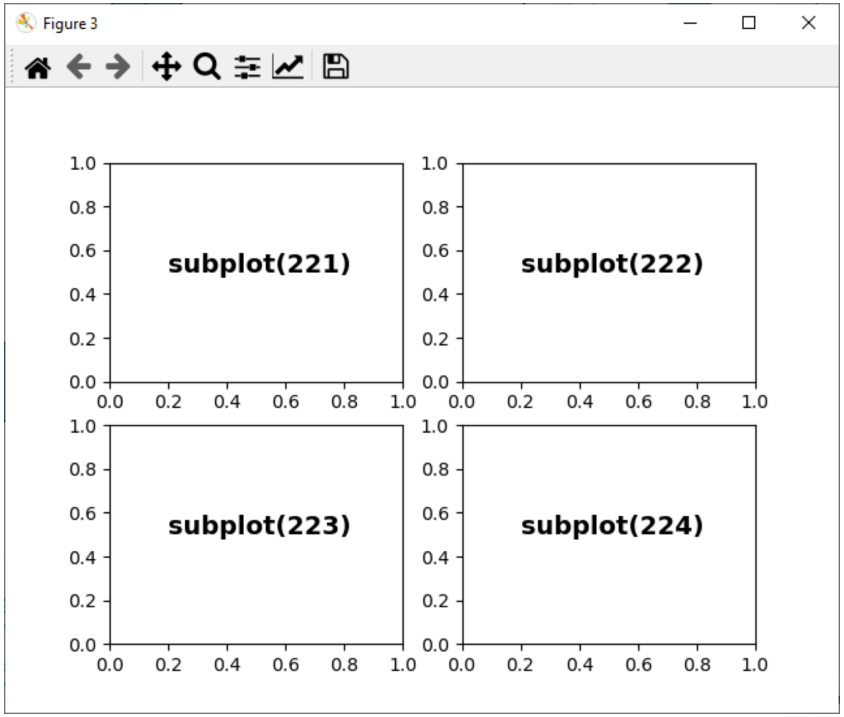




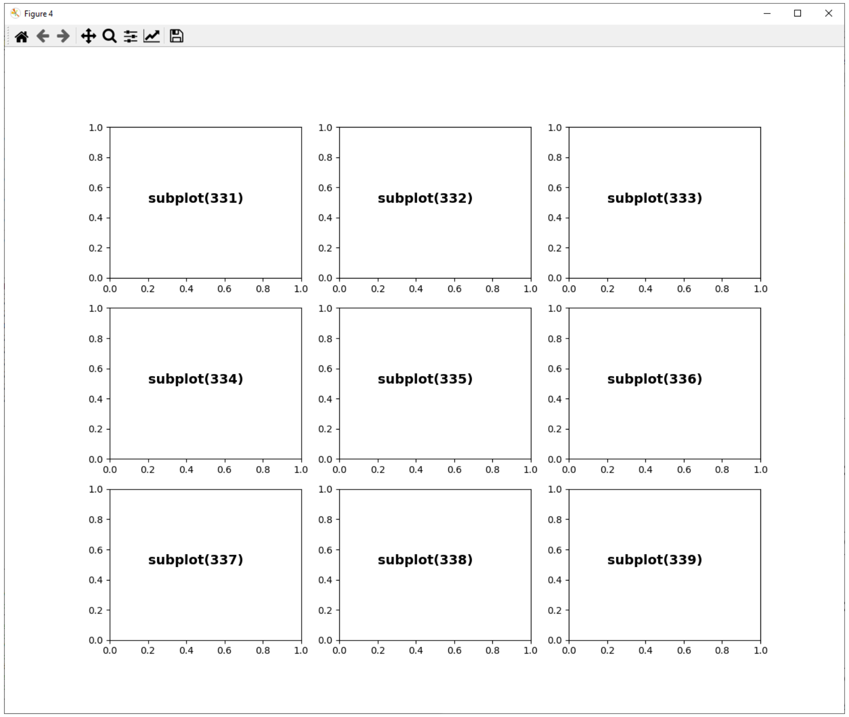
```
plt.figure(2); plt.clf()
ax1 = plt.subplot(121)
plt.text(0.2,0.5,'subplot(121)', fontsize=14, fontweight='bold')

ax2 = plt.subplot(122)
plt.text(0.2,0.5,'subplot(122)', fontsize=14, fontweight='bold')
```

Subplot Numbering

- 2 rows, 2 columns



- 3 rows, 3 columns



- Can have an arbitrary number of rows and columns

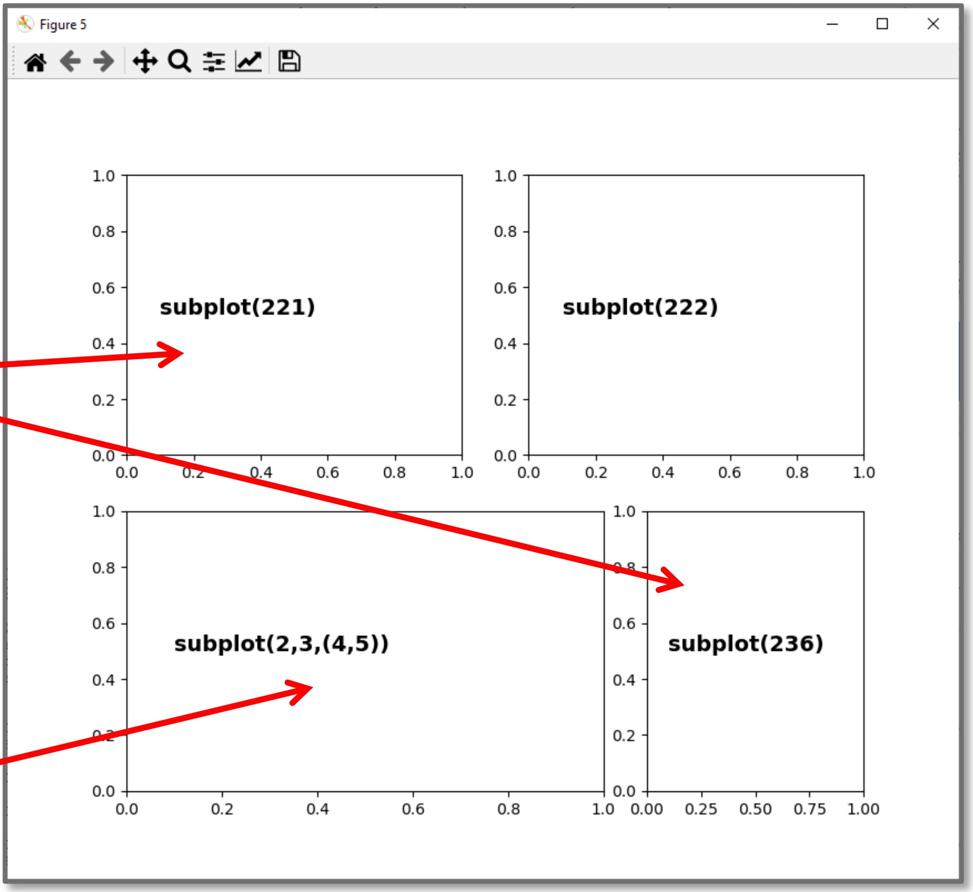
Subplot Numbering

```
subplot(m,n,p)
```

- m and n can vary within a figure window

- p can be specified as a range using a tuple:

```
subplot(m,n,(p1,p2))
```

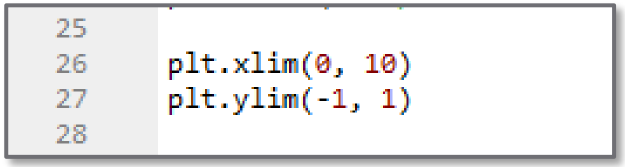


Axis Control

Axis Scaling – `plt.xlim()`, `plt.ylim()`

- □ Specify axis limits as positional arguments (`args`)

```
plt.xlim(xmin, xmax)
plt.ylim(ymin, ymax)
```



```
plt.xlim(0, 10)
plt.ylim(-1, 1)
```

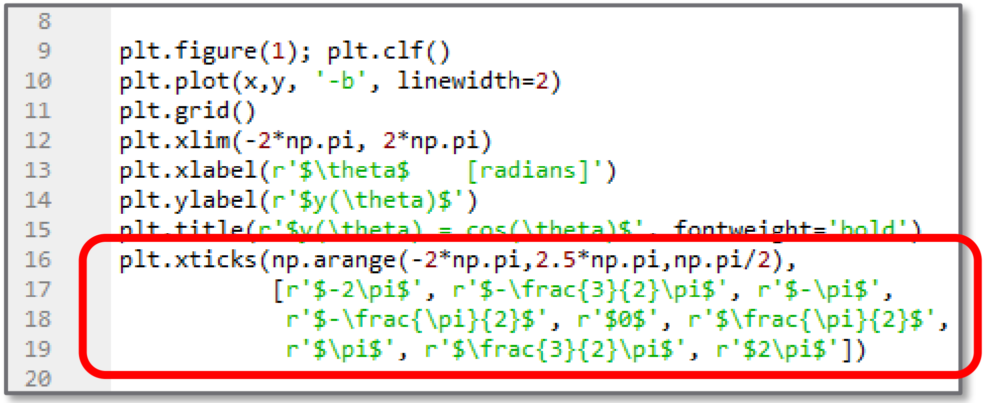
- □ Or, specify limits as keyword arguments (`kwargs`)

```
plt.xlim(left=xmin, right=xmax)
plt.ylim(bottom=ymin, top=ymax)
```

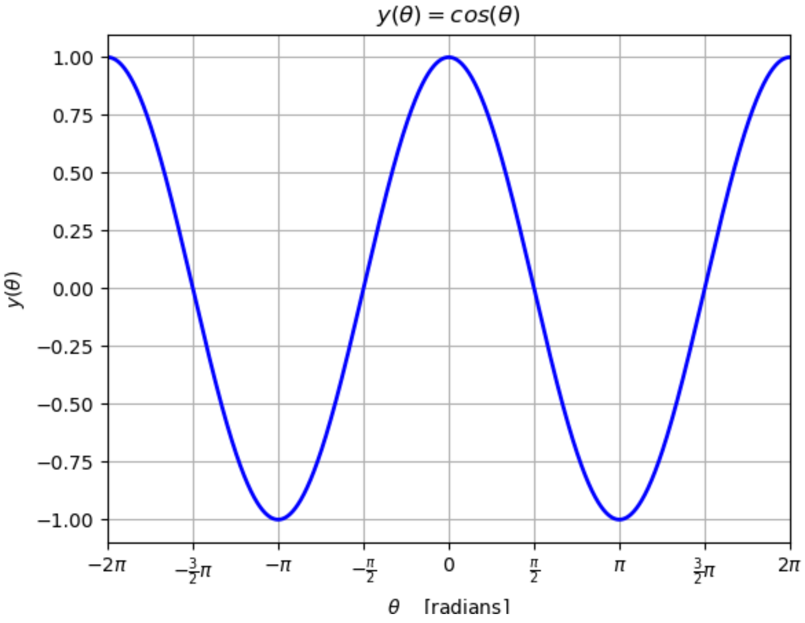
- □ Specify only one kwarg to allow for ***autoscaling*** of the other, e.g.:

```
plt.xlim(right=10e-3)
plt.ylim(bottom=-12)
```

Controlling Axis Tick Marks – plt.xticks()



```
plt.figure(1); plt.clf()
plt.plot(x,y, '-b', linewidth=2)
plt.grid()
plt.xlim(-2*np.pi, 2*np.pi)
plt.xlabel(r'$\theta$    [radians]')
plt.ylabel(r'$y(\theta)$')
plt.title(r'$y(\theta) = cos(\theta)$', fontweight='bold')
plt.xticks(np.arange(-2*np.pi,2.5*np.pi,np.pi/2),
           [r'$-2\pi$', r'$-\frac{3}{2}\pi$', r'$-\pi$',
            r'$-\frac{\pi}{2}$', r'$0$', r'$\frac{\pi}{2}$',
            r'$\pi$', r'$\frac{3}{2}\pi$', r'$2\pi$'])
```

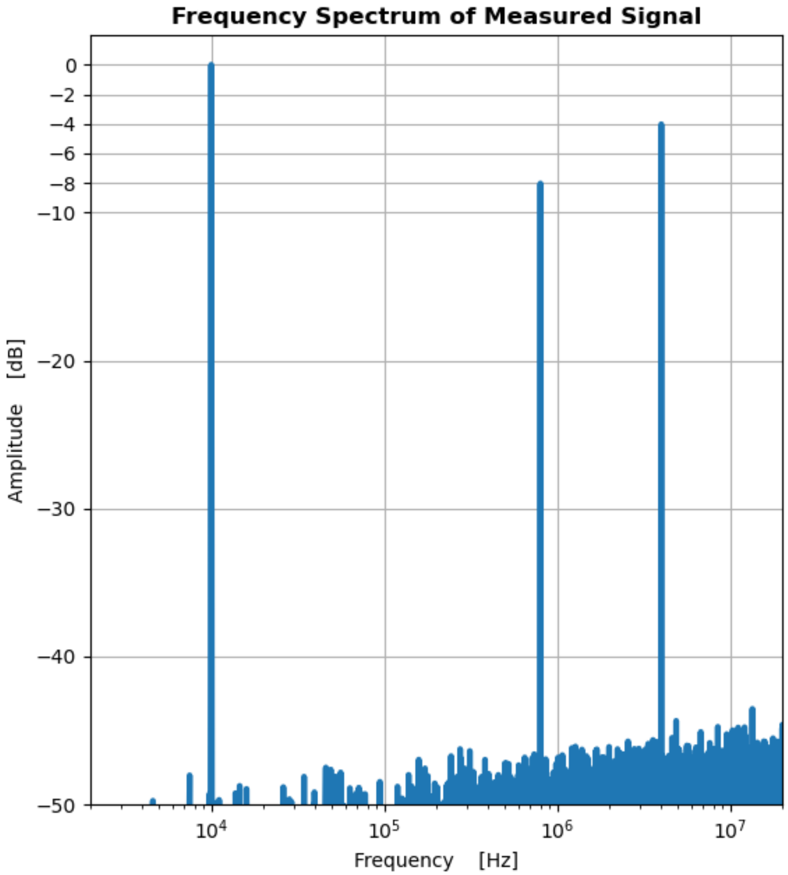


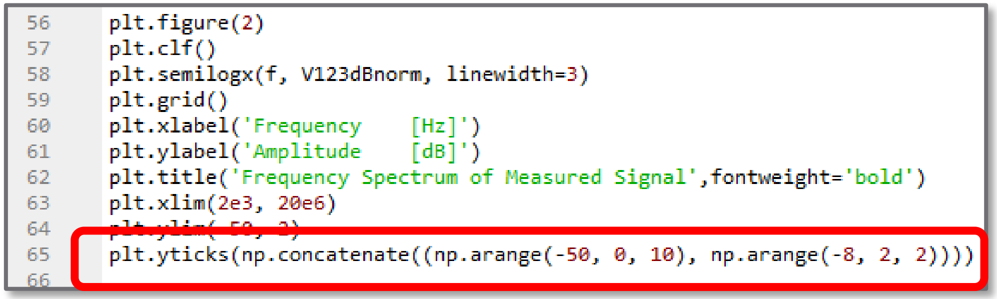
`plt.xticks(ticks, labels)`

- `ticks`: numeric tick locations
  - Array or list
- `labels`: text label at each tick
  - List of strings

- Interprets TeX characters
  - Raw strings – precede with r
  - Enclose in dollar signs, $

Controlling Axis Tick Marks – `plt.yticks()`





```
plt.figure(2)
plt.clf()
plt.semilogx(f, V123dBnorm, linewidth=3)
plt.grid()
plt.xlabel('Frequency    [Hz]')
plt.ylabel('Amplitude    [dB]')
plt.title('Frequency Spectrum of Measured Signal',fontweight='bold')
plt.xlim(2e3, 20e6)
plt.ylim(-50, 2)
plt.yticks(np.concatenate((np.arange(-50, 0, 10), np.arange(-8, 2, 2))))
```

- □ Non-uniform tick spacing is allowed
- □ If `labels` are not specified, default (numeric) labels are placed at each tick mark

Dual y-Axes – `plt.twinx()`

- Plot with ***different y-axes on right and left sides of figure***
  - Generate an axis object for the left-hand axis:

    ```
    ax1 = plt.axes()
    ```

  - Plot to the left-hand axis, generating a handle to the line for a legend creation:

    ```
    line1, = plt.plot(t2/1e-6,vs2,'-b', linewidth=2, label='$v_s(t)$')
    ```

  - Create a second (right-hand) y-axis, sharing a common x-axis:

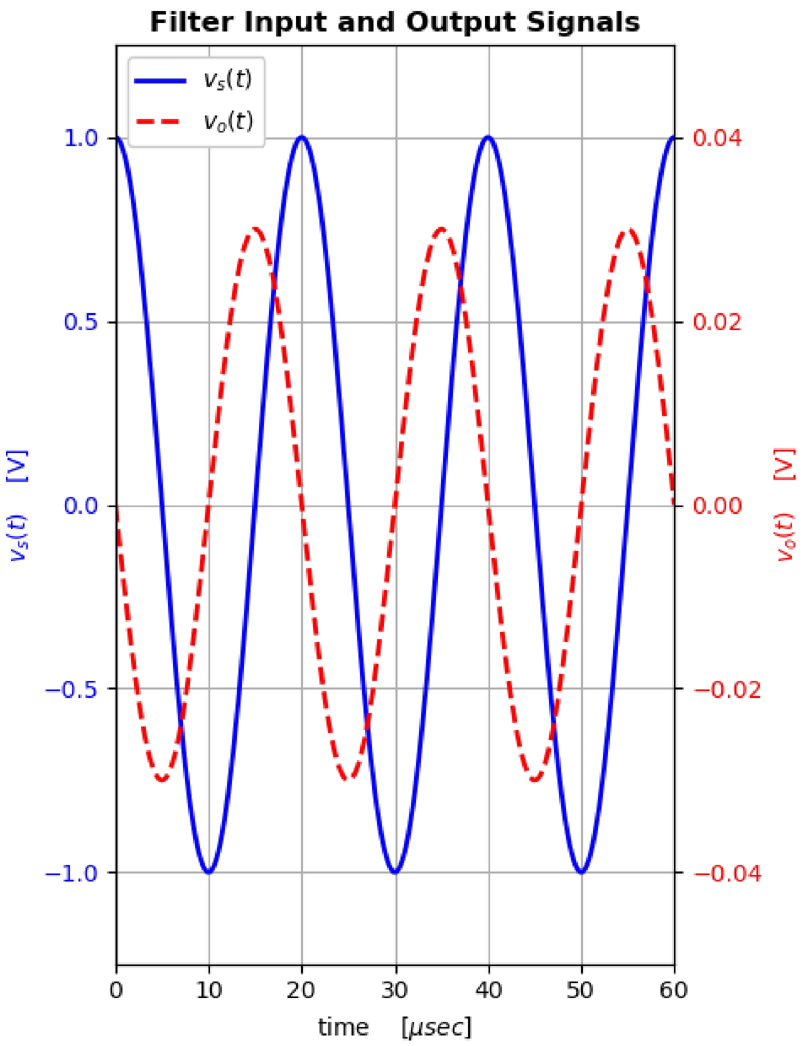
    ```
    ax2 = plt.twinx(ax1)
    ```

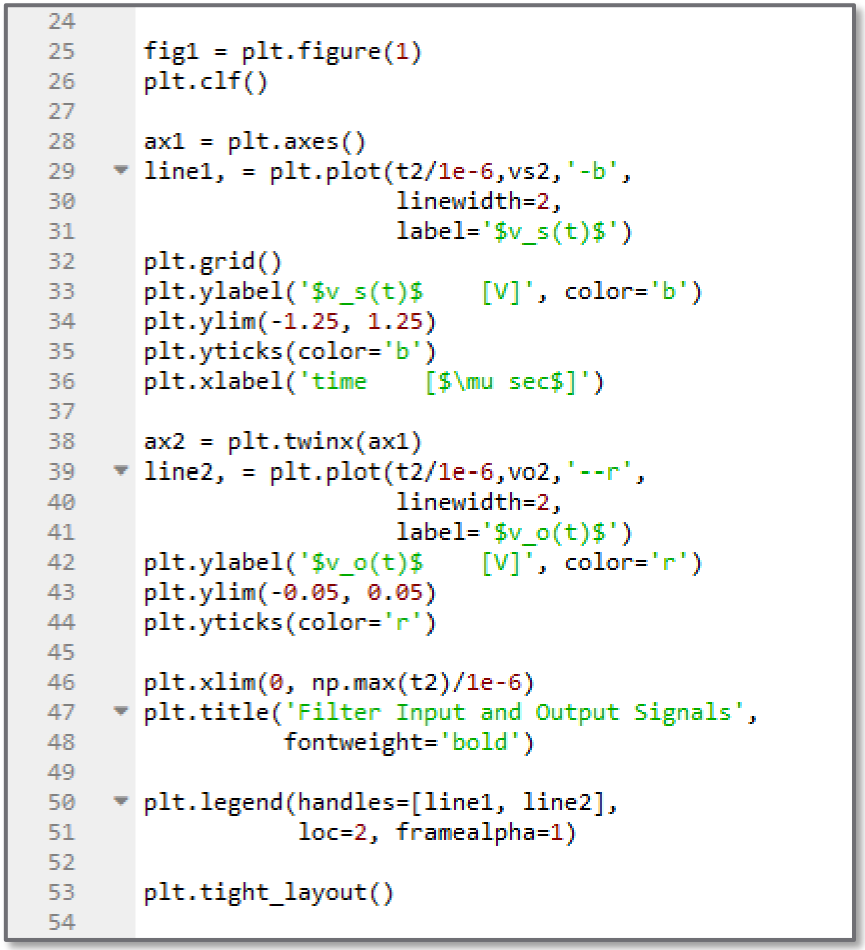
  - Plot to the right-hand axis:

    ```
    line2, = plt.plot(t2/1e-6,vo2,'--r', linewidth=2, label='$v_o(t)$')
    ```

- Use line handles to include all traces in a legend
- Can set colors of y-axis labels and ticks to match corresponding lines

Dual y-Axes – plt.twinx()





```

fig1 = plt.figure(1)
plt.clf()

ax1 = plt.axes()
line1, = plt.plot(t2/1e-6,vs2,'-b',
                  linewidth=2,
                  label='$v_s(t)$')
plt.grid()
plt.ylabel('$v_s(t)$    [V]', color='b')
plt.ylim(-1.25, 1.25)
plt.yticks(color='b')
plt.xlabel('time    [$\mu sec$]')

ax2 = plt.twinx(ax1)
line2, = plt.plot(t2/1e-6,vo2,'--r',
                  linewidth=2,
                  label='$v_o(t)$')
plt.ylabel('$v_o(t)$    [V]', color='r')
plt.ylim(-0.05, 0.05)
plt.yticks(color='r')

plt.xlim(0, np.max(t2)/1e-6)
plt.title('Filter Input and Output Signals',
          fontweight='bold')

plt.legend(handles=[line1, line2],
           loc=2, framealpha=1)

plt.tight_layout()

```

More Plot Types

Logarithmic Axes

- Useful for displaying datasets that span a very large range
- ***Log-log plot*** – both axes are logarithmic

```
plt.loglog(x, y, fmt, **kwargs)
```

- ***Logarithmic X-axis***

```
plt.semilogx(x, y, fmt, **kwargs)
```

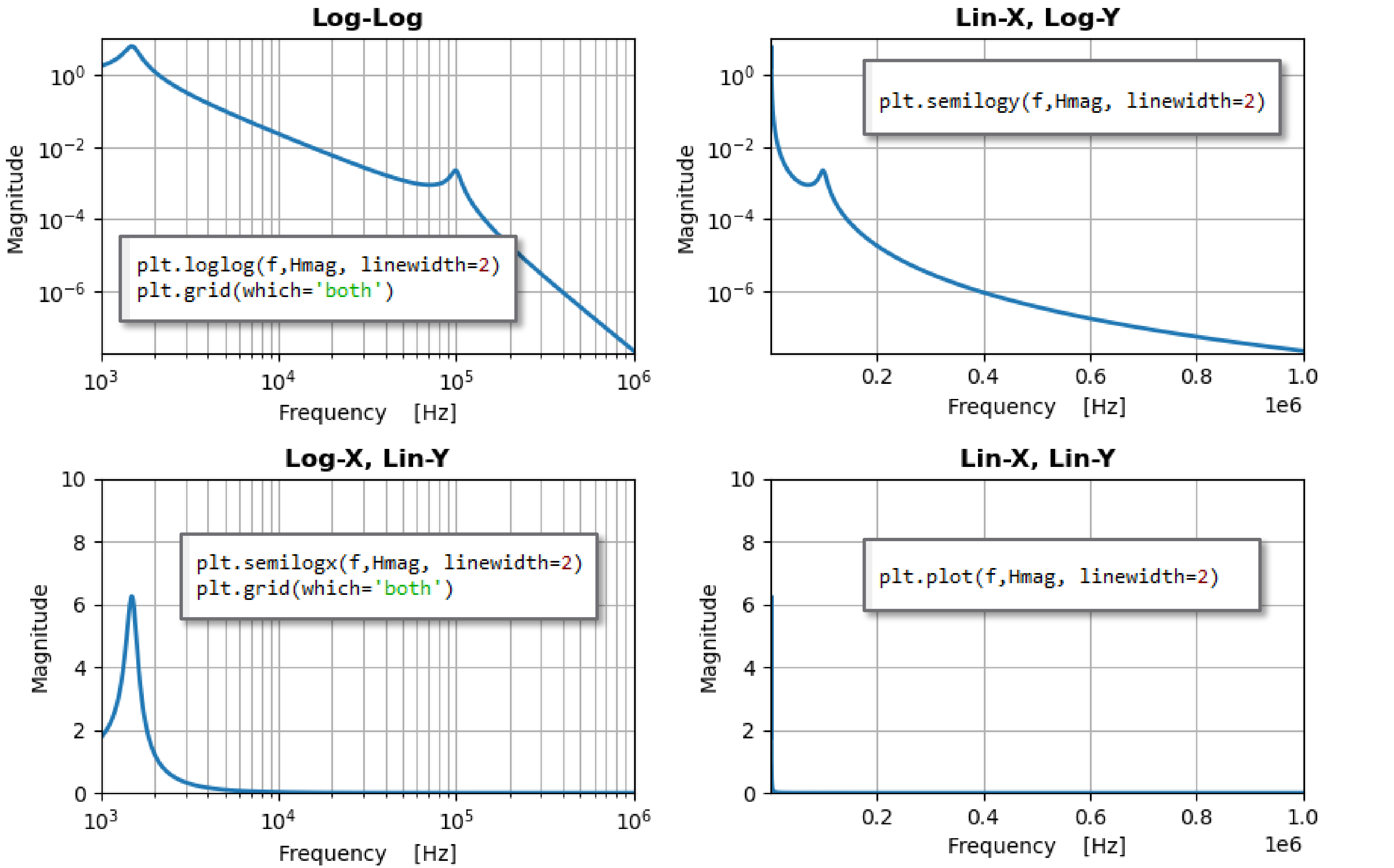
- ***Logarithmic Y-axis***

```
plt.semilogy(x, y, fmt, **kwargs)
```

- Generating ind. variable vector for log-x plots:

```
np.logspace(X1, X2, N)
```

Logarithmic Axes



```
plt.loglog(f,Hmag, linewidth=2)
plt.grid(which='both')
```

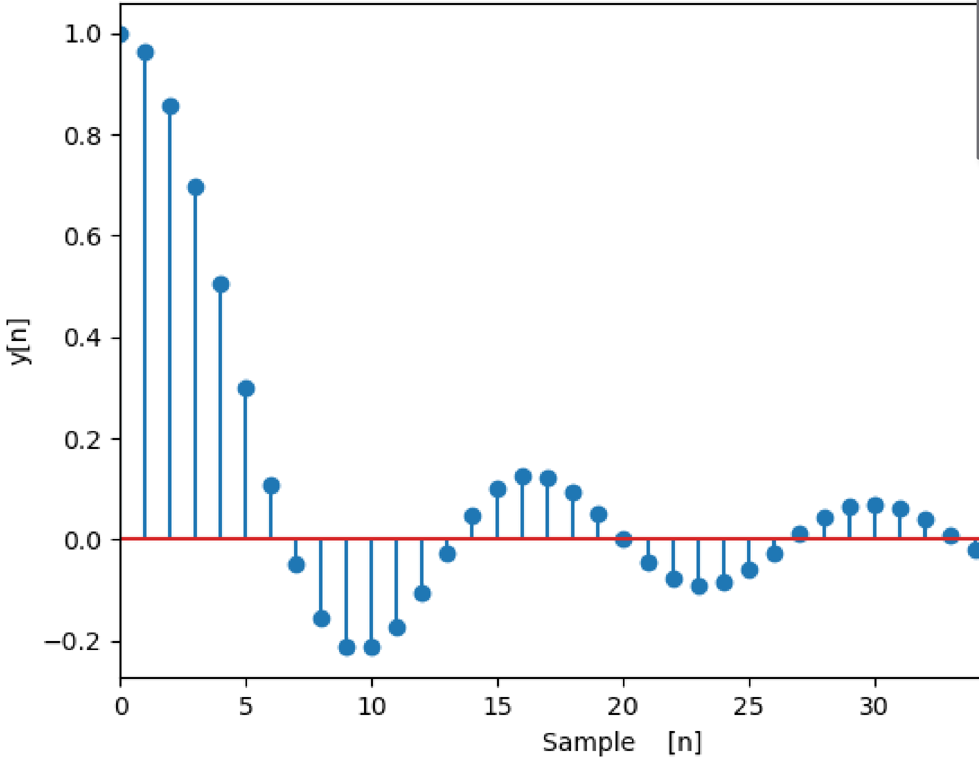
```
plt.semilogy(f,Hmag, linewidth=2)
```

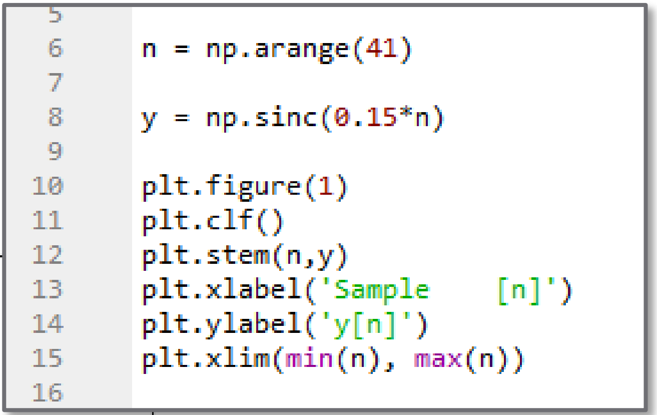
```
plt.semilogx(f,Hmag, linewidth=2)
plt.grid(which='both')
```

```
plt.plot(f,Hmag, linewidth=2)
```

Stem Plot – `plt.stem()`

```
plt.stem(x,y)
```



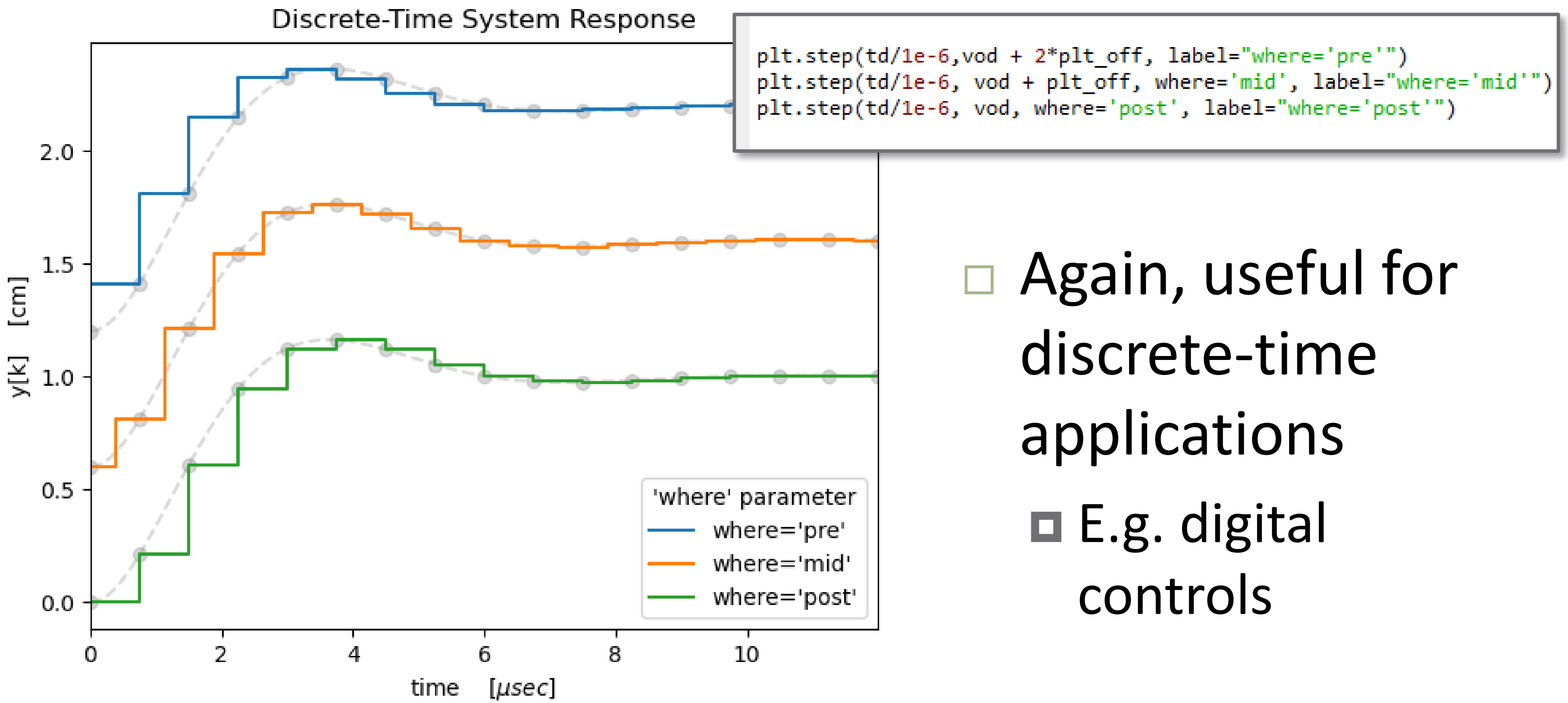


```
n = np.arange(41)

y = np.sinc(0.15*n)

plt.figure(1)
plt.clf()
plt.stem(n,y)
plt.xlabel('Sample    [n]')
plt.ylabel('y[n]')
plt.xlim(min(n), max(n))
```

- Good for plotting discrete-time data
  - E.g. digital control, signal processing applications

Plotting Zero-Order-Hold Data – `plt.step()`

```
plt.step(x, y, where='pre', **kwargs)
```

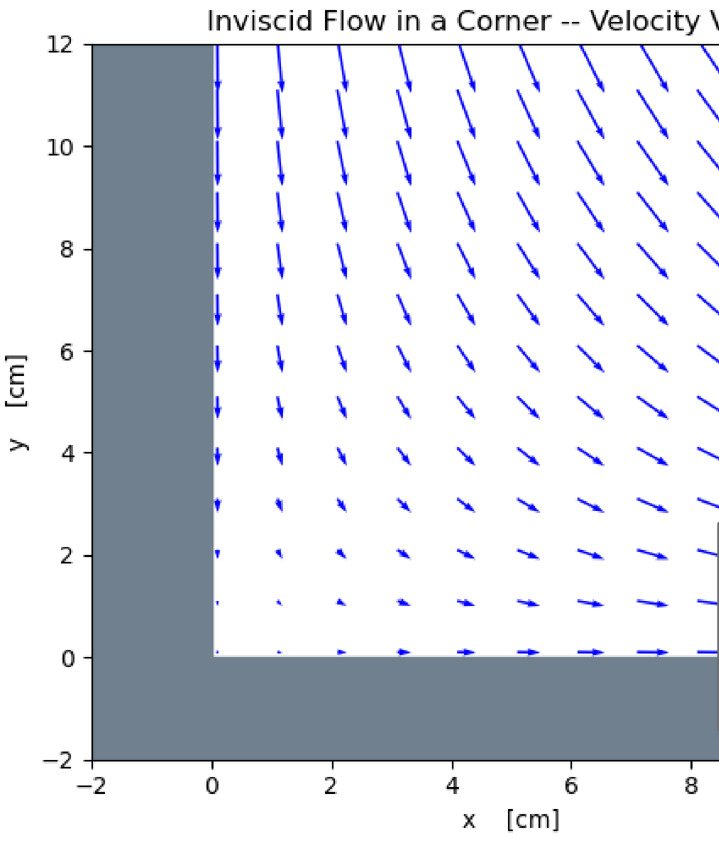


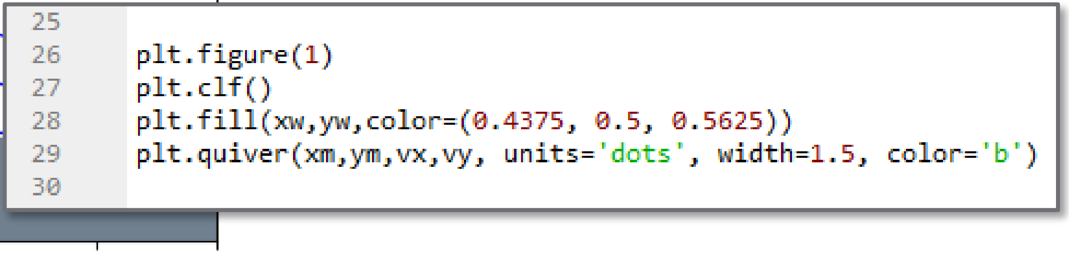
```
plt.step(td/1e-6,vod + 2*plt_off, label="where='pre'")
plt.step(td/1e-6, vod + plt_off, where='mid', label="where='mid'")
plt.step(td/1e-6, vod, where='post', label="where='post'")
```

- Again, useful for discrete-time applications
  - E.g. digital controls

Plotting Vector Fields – `plt.quiver()`

```
plt.quiver(x, y, u, v, **kwargs)
```



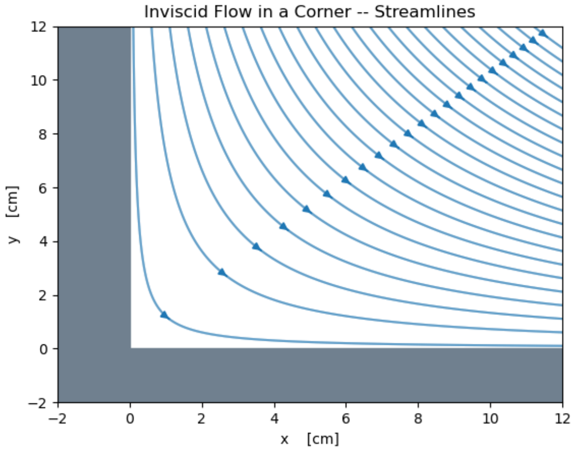


```
plt.figure(1)
plt.clf()
plt.fill(xw,yw,color=(0.4375, 0.5, 0.5625))
plt.quiver(xm,ym,vx,vy, units='dots', width=1.5, color='b')
```

- □ `x,y`: matrices of x,y coordinates – generate with `np.meshgrid()` – more later
- □ `u,v`: velocity components at `x,y` locations – matrices

Streamline Plots – plt.streamplot()

```
plt.streamplot(x, y, u, v,
            density=den,
            start_points=start)
```

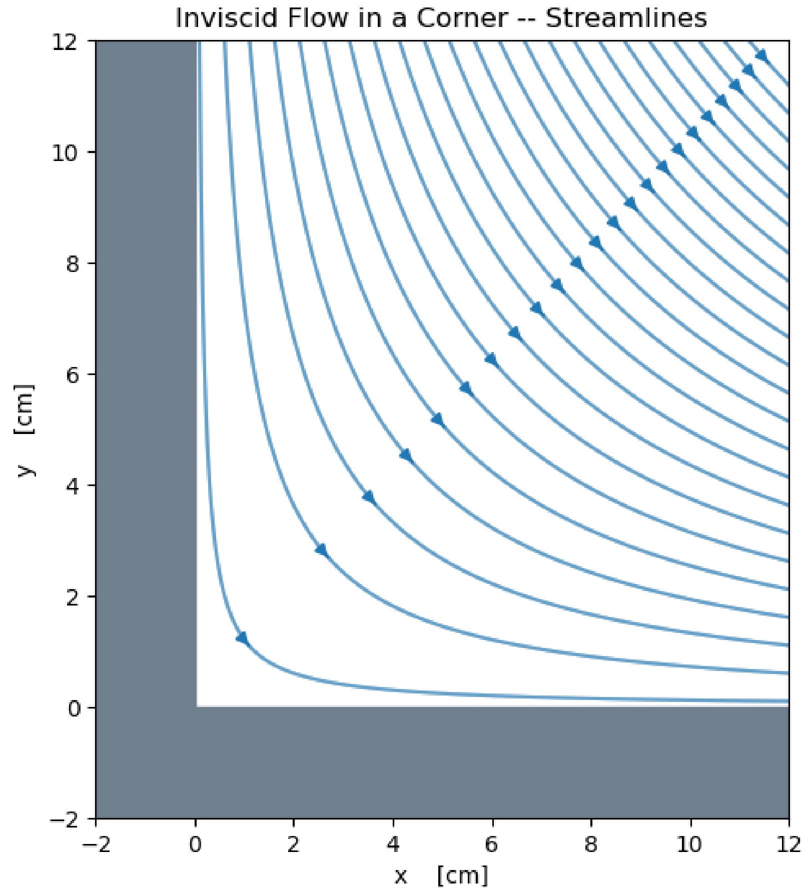


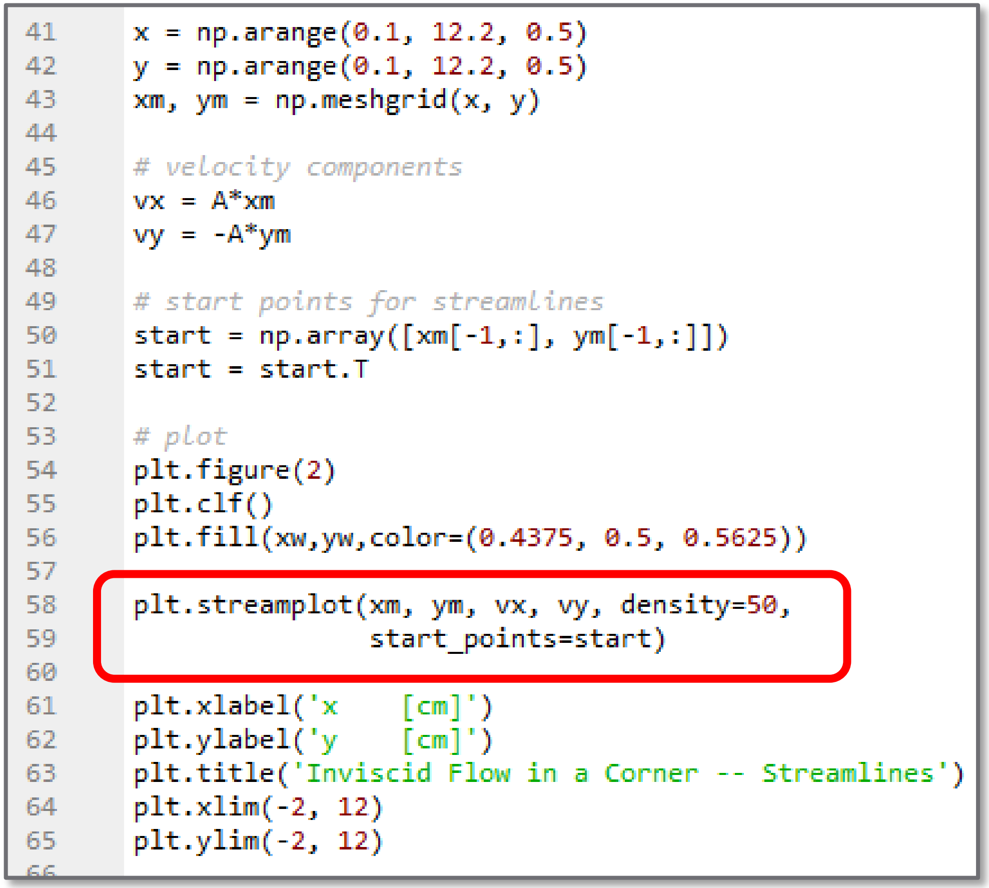
  - x,y,u,v: same as for plt.quiver()
  - den: *optional* - allowable density of streamlines – default: 1 – max. 30 lines in each direction
  - start: 2 x N matrix of (x, y) starting coordinates for streamlines

- Streamlines will break to avoid exceeding density setting
  - Set density excessively high, e.g. density=30
  - Specify start_points to control number of gridlines, but have them be unbroken

Streamline Plots – plt.streamplot()

```
plt.streamplot(x, y, u, v, density=den, start_points=start)
```



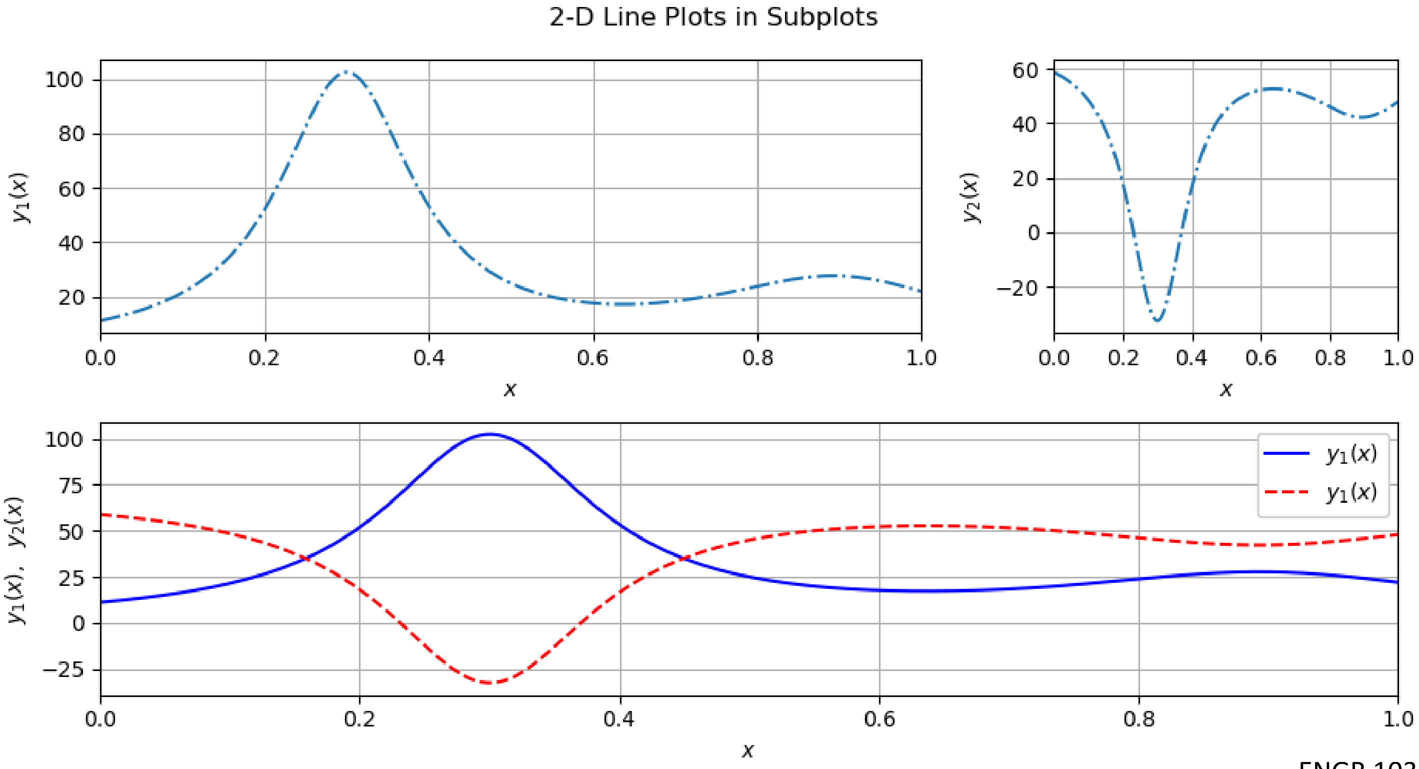


```
x = np.arange(0.1, 12.2, 0.5)
y = np.arange(0.1, 12.2, 0.5)
xm, ym = np.meshgrid(x, y)

# velocity components
vx = A*xm
vy = -A*ym

# start points for streamlines
start = np.array([xm[-1,:], ym[-1,:]])
start = start.T

# plot
plt.figure(2)
plt.clf()
plt.fill(xw,yw,color=(0.4375, 0.5, 0.5625))

plt.streamplot(xm, ym, vx, vy, density=50,
               start_points=start)

plt.xlabel('x    [cm]')
plt.ylabel('y    [cm]')
plt.title('Inviscid Flow in a Corner -- Streamlines')
plt.xlim(-2, 12)
plt.ylim(-2, 12)
```

Subplots

Exercise

- Save your script from the previous exercise to a new file
- Modify the script to produce the following figure
  - You may find the following useful:
    - `plt.suptitle(`*`titlestr`*`)`
    - `plt.tight_layout()`


2-D Line Plots in Subplots
$y_1(x)$
x
$y_2(x)$
x
$y_1(x),\ y_2(x)$
x
$y_1(x)$
$y_1(x)$