HAB 619 Introduction to Scientific Computing in Sports Science

De Motu Animalium G.Borelli (1680)

#5

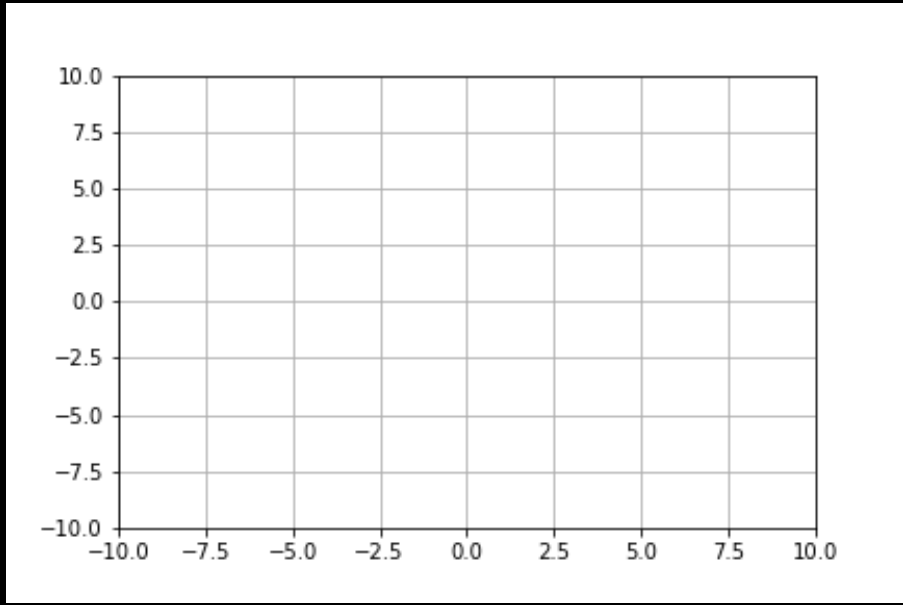
SERDAR ARITAN

serdar.aritan@hacettepe.edu.tr

Biyomekanik Araştırma Grubu
www.biomech.hacettepe.edu.tr
Spor Bilimleri Fakültesi
www.sbt.hacettepe.edu.tr
Hacettepe Universitesi, Ankara, Türkiye
www.hacettepe.edu.tr

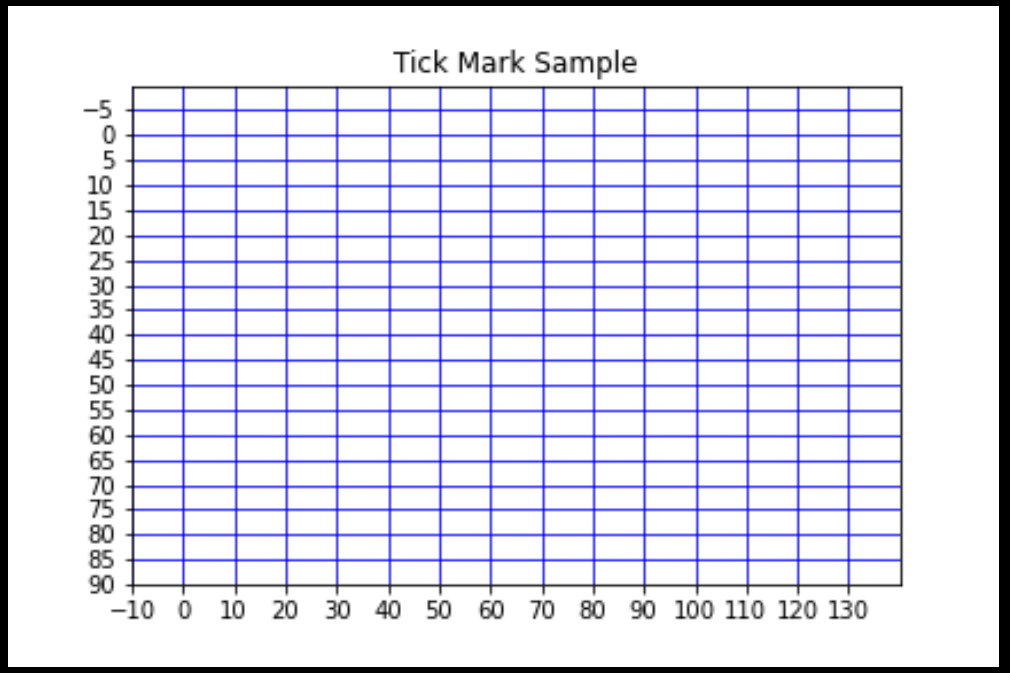
Establishing the Size of the Plotting Area

```
import numpy as np
import matplotlib.pyplot as plt

x1=-10
x2=10
y1=-10
y2=10
plt.axis([x1,x2,y1,y2])

plt.axis('on')
plt.grid(True)

plt.show()
```



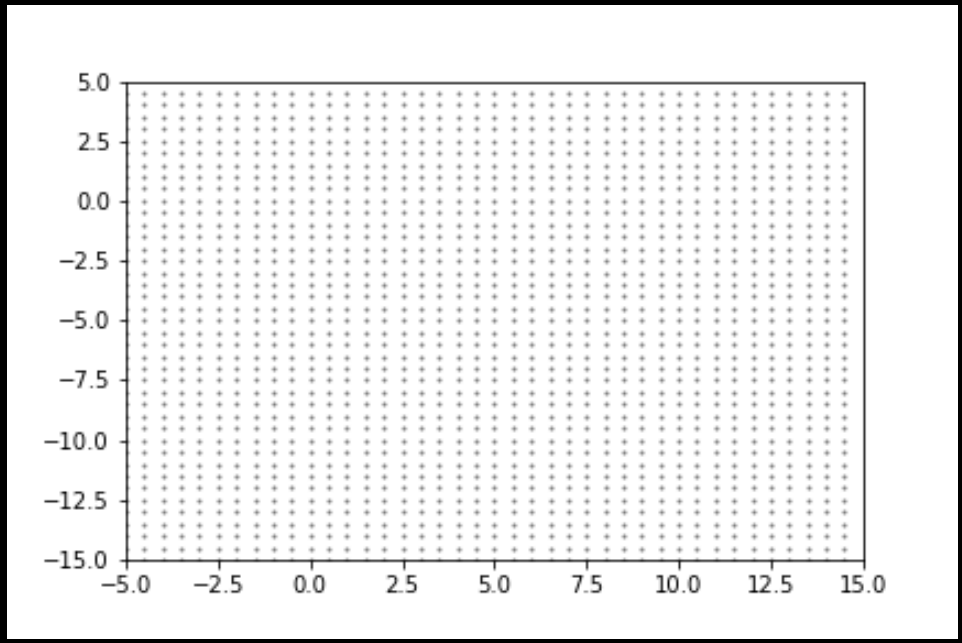
Tick Marks

```
import numpy as np
import matplotlib.pyplot as plt

x1 = -10
x2 = 140
y1 = 90
y2 = -10
plt.axis([x1,x2,y1,y2])
plt.axis('on')
plt.grid(True, color='b')
plt.title('Tick Mark Sample')
xmin = x1
xmax = x2
dx = 10
ymin = y1
ymax = y2
dy = -5
plt.xticks(np.arange(xmin, xmax, dx))
plt.yticks(np.arange(ymin, ymax, dy))

plt.show()
```

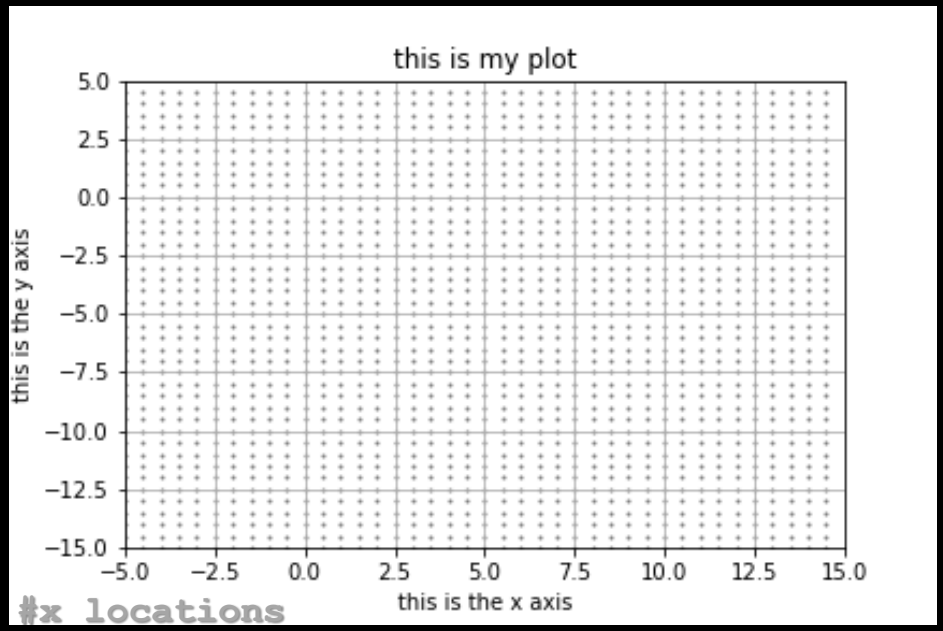


Tick Marks

```
import numpy as np
import matplotlib.pyplot as plt

x1=-5
x2=15
y1=-15
y2=5
plt.axis([x1,x2,y1,y2])
plt.axis('on')

dx=.5 #x spacing
dy=.5 #y spacing
for x in np.arange(x1,x2,dx):
    for y in np.arange(y1,y2,dy):  #y locations
        plt.scatter(x,y,s=1,color='grey') #plot a grey point at x,y

plt.show()
```



Plot Title and Labelling the Axes




```
import numpy as np
import matplotlib.pyplot as plt

x1=-5
x2=15
y1=-15
y2=5
plt.axis([x1,x2,y1,y2])
plt.axis('on')
dx=.5 #x spacing
dy=.5 #y spacing
for x in np.arange(x1,x2,dx):
    for y in np.arange(y1,y2,dy):  #y locations
        plt.scatter(x,y,s=1,color='grey') #plot a grey point at x,y

plt.grid(True)
plt.xlabel('this is the x axis')
plt.ylabel('this is the y axis')
plt.title('this is my plot')

plt.show()
```

Color

'k' for black
'b' for blue
'c' for cyan
'g' for green
'm' for magenta
'r' for red
'y' for yellow
'gray' or 'grey'
'lightgray' or 'lightgrey'

For example, the following statement will plot a green dot at coordinates x,y:

```
plt.scatter(x, y, color = 'g')
```

NumPy

Color

https://matplotlib.org/examples/color/named_colors.html

Color

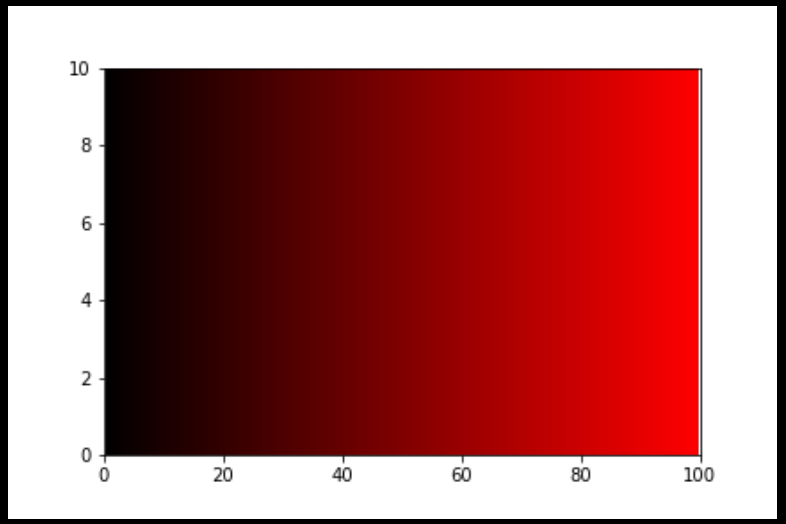
https://matplotlib.org/examples/color/named_colors.html

```
"""
========================
Visualizing named colors
========================

Simple plot example with the named colors and its visual representation.
"""
from __future__ import division

import matplotlib.pyplot as plt
from matplotlib import colors as mcolors


colors = dict(mcolors.BASE_COLORS, **mcolors.CSS4_COLORS)

# Sort colors by hue, saturation, value and name.
by_hsv = sorted((tuple(mcolors.rgb_to_hsv(mcolors.to_rgba(color)[:3])), name)
                for name, color in colors.items())
sorted_names = [name for hsv, name in by_hsv]
……..
……..
ax.set_xlim(0, X)
ax.set_ylim(0, Y)
ax.set_axis_off()

fig.subplots_adjust(left=0, right=1,
                    top=1, bottom=0,
                    hspace=0, wspace=0)
plt.show()
```

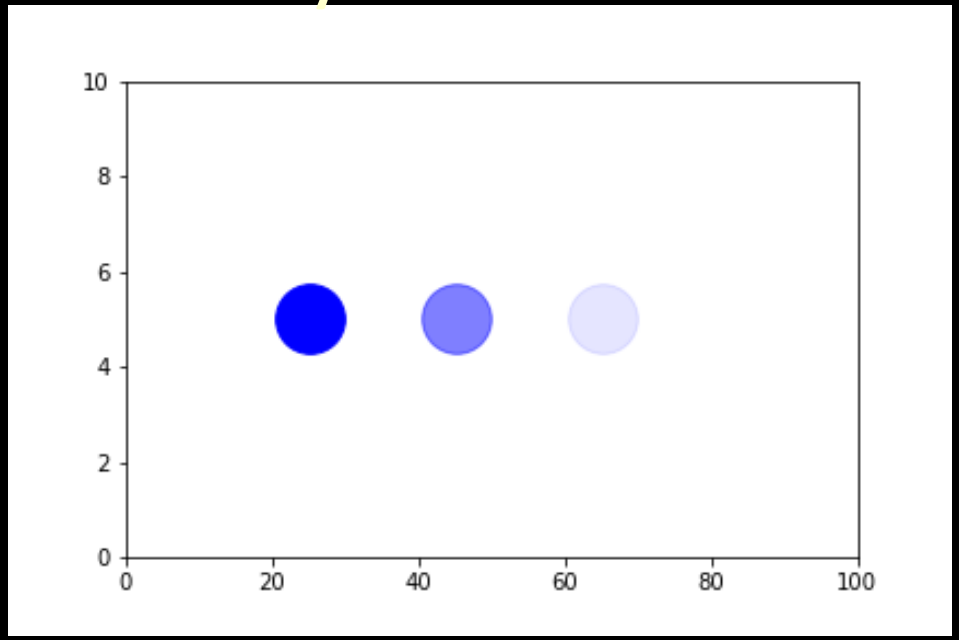
Color Mixing

You can mix your own hues from the primary colors of red (r), green (g), and blue (b) with the specification `color=(r,g,b)` where r,g,b are the values of red, green, and blue in the mix, with values of each ranging from 0 to 1.

```
import numpy as np
import matplotlib.pyplot as plt



plt.axis([0,100,0,10])

for x in np.arange(1,100,1):
    r=x/100
    g=0
    b=0
    plt.plot([x,x],[0,10],linewidth=5,color=(r,g,b))
plt.show()

# r = x/100, g = 0, b = x/100
# r = x/100, g = x/100, b = 0
```



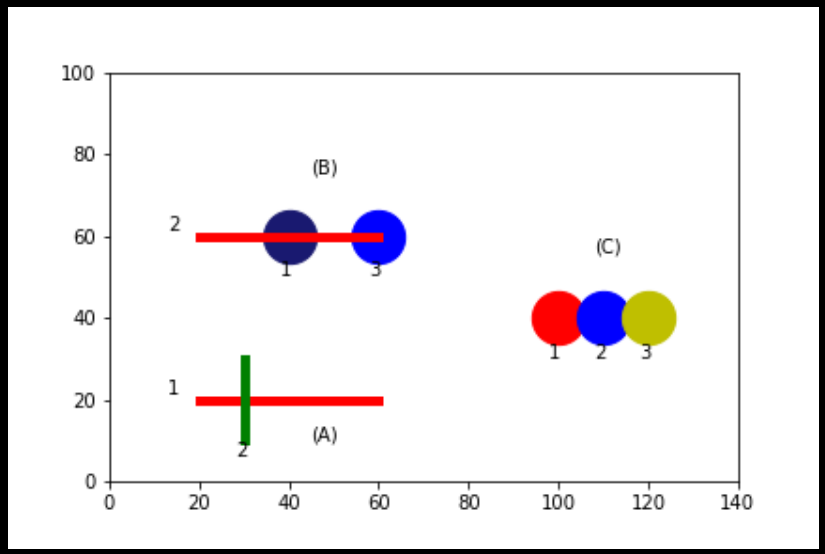
Color Intensity




```
import numpy as np
import matplotlib.pyplot as plt


plt.axis([0,100,0,10])


plt.scatter(25, 5, s=1000, color='b', alpha=1)
plt.scatter(45, 5, s=1000, color='b', alpha=.5)
plt.scatter(65, 5, s=1000, color='b', alpha=.1)


plt.show()
```

Overplotting

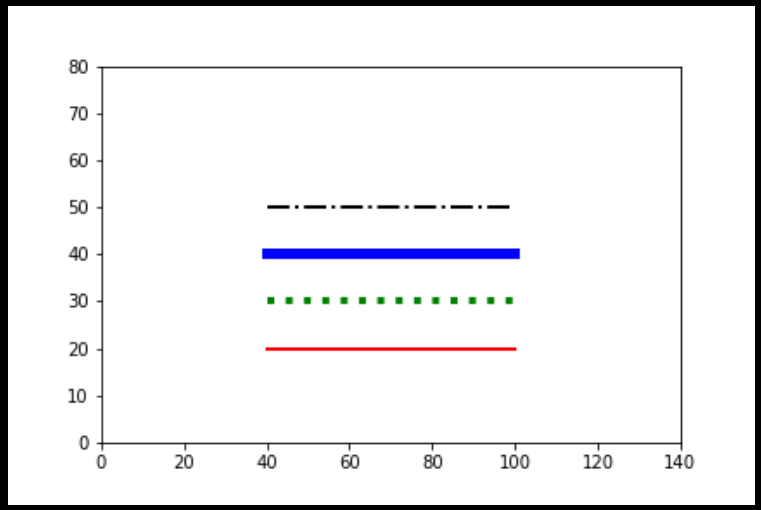
```
import numpy as np
import matplotlib.pyplot as plt
plt.axis([0,140,0,100])
#------------------------- A
plt.text(45,10,'(A)')
plt.plot([20,60],[20,20],linewidth=5,color='r')
plt.text(13,21,'1')
plt.plot([30,30],[10,30],linewidth=5,color='g')
plt.text(28,6,'2')
#------------------------- B
plt.text(45,75,'(B)')
plt.scatter(40,60,s=800,color='midnightblue')
plt.text(38,50,'1')
plt.plot([20,60],[60,60],linewidth=5,color='r')
plt.text(13,61,'2')
plt.scatter(60,60,s=800,color='b')
plt.text(58,50,'3')
#------------------------- C
plt.text(108,56,'(C)')
plt.scatter(100,40,s=800,color='r')
plt.text(98,30,'1')
plt.scatter(110,40,s=800,color='b')
plt.text(108,30,'2')
plt.scatter(120,40,s=800,color='y')
plt.text(118,30,'3')

plt.show()
```



Lines Using `plot()`

`plt.plot([x1,x2],[y1,y2],linewidth=linewidth, color='color' ,linestyle='linestyle')`

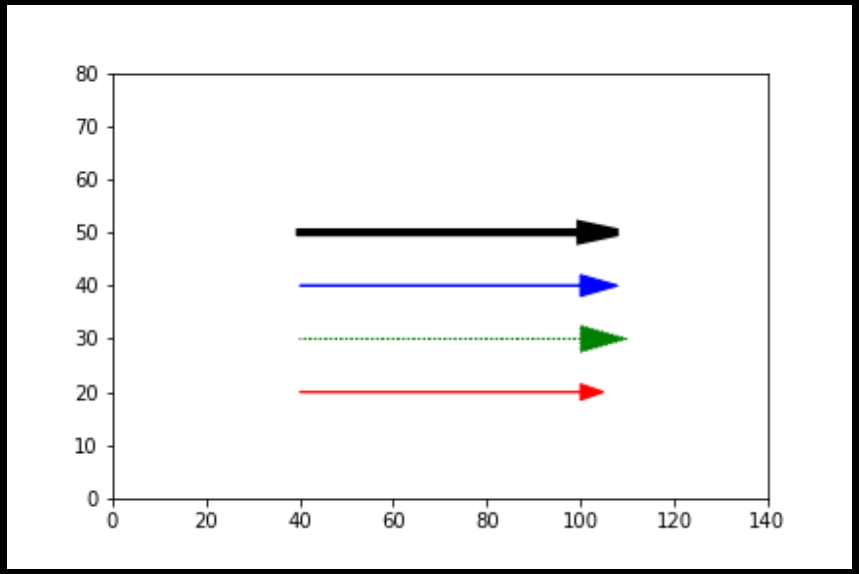
This command draws a line from `x1,y1` to `x2,y2`. It has a width specified by `linewidth`, a color by color, and a style by `linestyle`.




```
import numpy as np
import matplotlib.pyplot as plt


plt.axis([0,140,0,80])

plt.plot([40,100], [20,20], linewidth=2, color='r')
plt.plot([40,100], [30,30], linewidth=4, color='g', linestyle= ':')
plt.plot([40,100], [40,40], linewidth=6, color='b', linestyle= '-')
plt.plot([40,100], [50,50], linewidth=2, color='k', linestyle= '-.')

plt.show()
```

Arrows

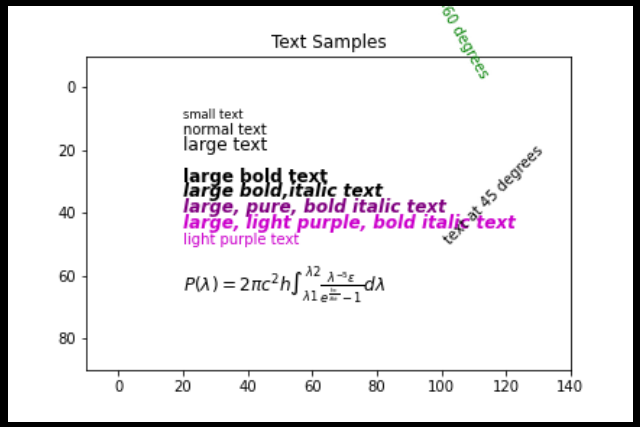
```
plt.arrow(x, y, Δx, Δy, line_width='linewidth', head_length = 'headlength',
 head_width = 'headwidth', color = 'color' ,linestyle = 'linestyle')
```




```
import numpy as np
import matplotlib.pyplot as plt


plt.axis([0,140,0,80])

plt.arrow(40,20,60,0, linewidth=1, color='r', head_length=5, head_width=3)
plt.arrow(40,30,60,0, linewidth=1, color='g', linestyle=':', head_length=10, head_width=5)
plt.arrow(40,40,60,0, linewidth=1, color='b', linestyle='-', head_length=8, head_width=4)
plt.arrow(40,50,60,0, linewidth=4, color='k', linestyle='-', head_length=8, head_width=3)

plt.show()
```

Text

```
plt.text(x,y, 'text', color='color', size='size', fontweight='fontweight',
 fontstyle='fontstyle', rotation=degrees)
```




```
import numpy as np
import matplotlib.pyplot as plt

plt.axis([-10,140,90,-10])
plt.axis('on')
plt.grid(False)
plt.title('Text Samples')
plt.text(20,10,'small text',size='small')
plt.text(20,15,'normal text')
plt.text(20,20,'large text',size='large')
plt.text(20,30,'large bold text',size='large',fontweight='bold')
plt.text(20,35,'large bold,italic text',size='large',\
         fontweight='bold', fontstyle='italic')
plt.text(20,40,'large, pure, bold italic text',size='large',\
         fontweight='bold',fontstyle='italic',color=(.5,0,.5))
plt.text(20,45,'large, light purple, bold italic text',size='large',\
         fontweight='bold',fontstyle='italic',color=(.8,0,.8))
plt.text(20,50,'light purple text',color=(.8,0,.8))
plt.text(100,50,'text at 45 degrees',rotation=45,color='k')
plt.text(90,-3,'text at -60 degrees',rotation=-60,color='g')
plt.text(20,65,r'$P(\lambda)=2 \pi c^{2} h\int_{\lambda1}^{\lambda2}\frac{\lambda^{-
5}\epsilon}{e^{\frac{hc}{\lambda k t}}-1}d\lambda$',size='large')
plt.show()
```

Advanced Matplotlib Concepts

rcParams

It contains all Matplotlib settings that are used to create the default styles of the figures. You import it directly from the matplotlib namespace:

```
>>>from matplotlib import rcParams
>>> rcParams
...
'axes.grid': False,
'axes.grid.axis': 'both',
'axes.grid.which': 'major',
'axes.labelcolor': 'black',
'axes.labelpad': 4.0,
'axes.labelsize': 'medium',
'axes.labelweight': 'normal',
'axes.linewidth': 0.8,
...

rcParams['figure.figsize'] = 8, 6
rcParams['legend.fontsize'] = "large"
rcParams['xtick.major.size'] = 4
rcParams['xtick.minor.size'] = 1
```

Advanced Matplotlib Concepts

Get_* functions

The key to a great plot is customization. To achieve this, you should know how to extract the components you want to customize.

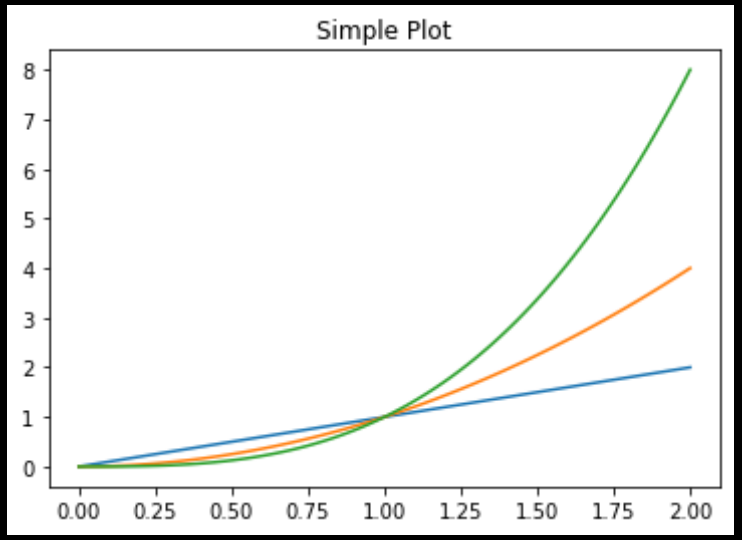
```
>>> import matplotlib.pyplot as plt
>>> fig, ax = plt.subplots()
>>> [func for func in dir(ax) if func.startswith("get")]
['get_adjustable',
 'get_agg_filter',
 'get_alpha',
 'get_anchor',
 'get_animated',
 'get_aspect',
 'get_autoscale_on',
 'get_autoscalex_on',
```

Advanced Matplotlib Concepts

Get_* functions

If you want to customize the xticks of this plot

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)

fig, ax = plt.subplots()  # Create a figure and an axes.

ax.plot(x, x, label="linear")
ax.plot(x, x ** 2, label="quadratic")
ax.plot(x, x ** 3, label="cubic")

ax.set_title("Simple Plot")

plt.show()
```

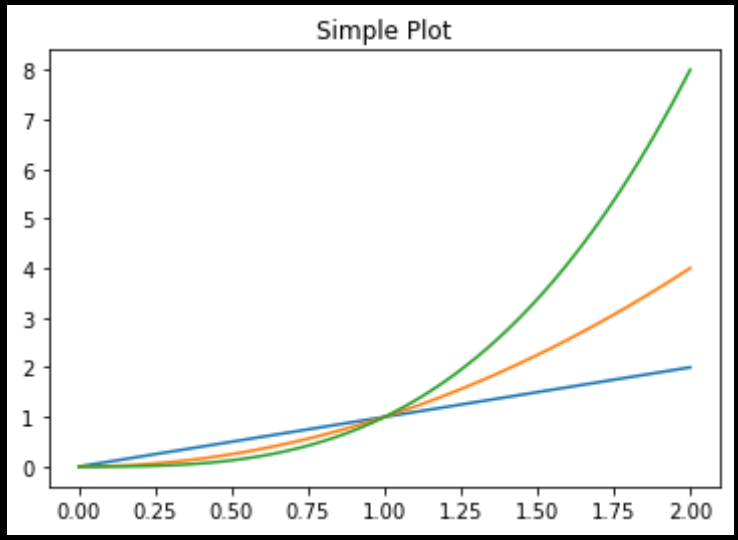


Advanced Matplotlib Concepts

Get_* functions

If you want to customize the xticks of this plot

```
>>> ax.get_xticklabels()

[Text(-0.25, 0, '−0.25'),
 Text(0.0, 0, '0.00'),
 Text(0.25, 0, '0.25'),
 Text(0.5, 0, '0.50'),
 Text(0.75, 0, '0.75'),
 Text(1.0, 0, '1.00'),
 Text(1.25, 0, '1.25'),
 Text(1.5, 0, '1.50'),
 Text(1.75, 0, '1.75'),
 Text(2.0, 0, '2.00'),
 Text(2.25, 0, '2.25')]
```

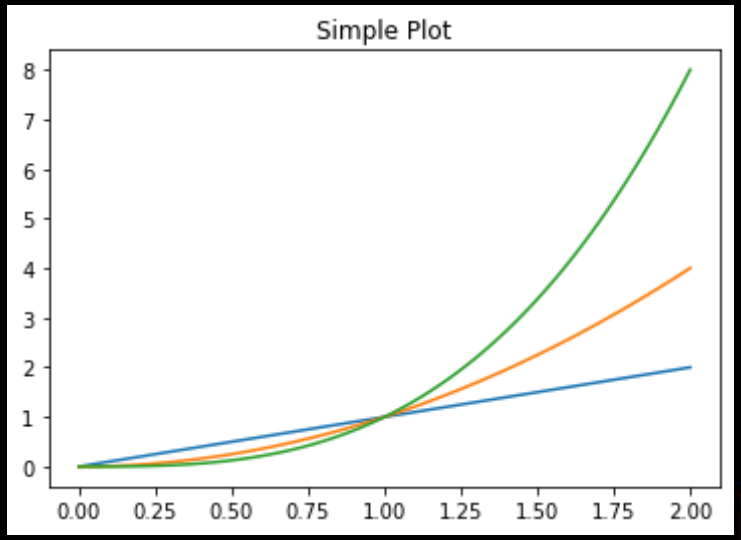



```
>>> ax.get_xticks()
array([-0.25,  0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,
        2.  ,  2.25])
```

Advanced Matplotlib Concepts

getp / setp

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)

fig, ax = plt.subplots()  # Create a figure and an axes.

l1 = ax.plot(x, x, label="linear")
l2 = ax.plot(x, x ** 2, label="quadratic")
l3 = ax.plot(x, x ** 3, label="cubic")

ax.set_title("Simple Plot")

plt.show()
```



Advanced Matplotlib Concepts

getp / setp

```
plt.getp(l2)
    agg_filter = None
    alpha = None
    animated = False
    antialiased or aa = True
    .
    .
    .
    rasterized = False
    sketch_params = None
    snap = None
    solid_capstyle = projecting
    solid_joinstyle = round
    transform = CompositeGenericTransform(     TransformWrapper(   ...
    transformed_clip_path_and_affine = (None, None)
    url = None
    visible = True
    xdata = [0.          0.02020202 0.04040404 0.06060606 0.080...
    xydata = [[0.          0.         ]  [0.02020202 0.00040812] ...
    ydata = [0.          0.00040812 0.00163249 0.00367309 0.006...
    zorder = 2
```

Advanced Matplotlib Concepts

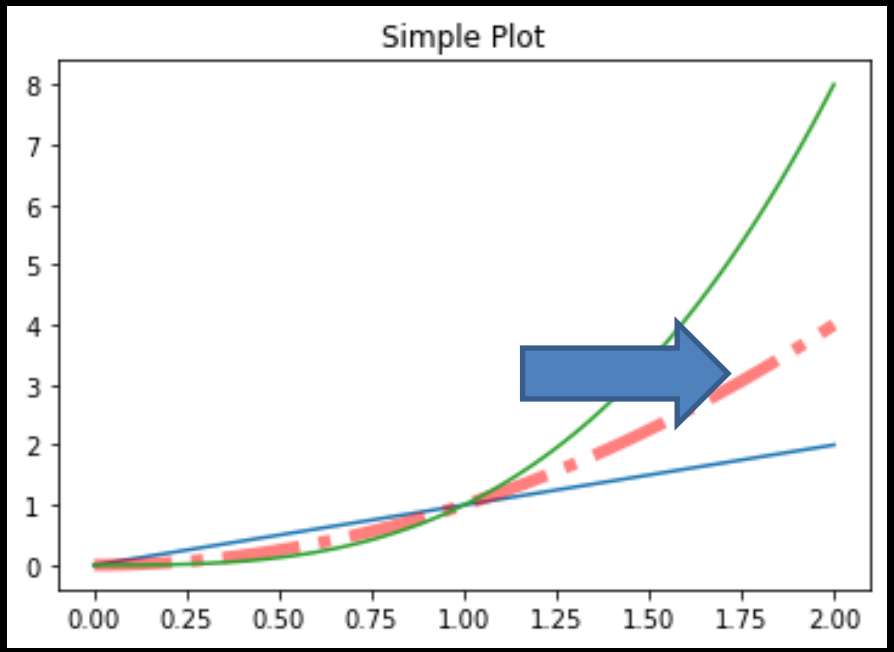
getp / setp

```
plt.setp(l2)
  agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi
value, and returns a (m, n, 3) array
  alpha: scalar or None
  animated: bool
  antialiased or aa: bool
  clip_box: `.Bbox`
  clip_on: bool
  clip_path: Patch or (Path, Transform) or None
  color or c: color
  label: object
  linestyle or ls: {'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
  linewidth or lw: float
  marker: marker style string, `~.path.Path` or `~.markers.MarkerStyle`
  markeredgecolor or mec: color
  markeredgewidth or mew: float
  markerfacecolor or mfc: color
  markerfacecoloralt or mfcalt: color
  markersize or ms: float
  markevery: None or int or (int, int) or slice or list[int] or float or (float,
float) or list[bool]
  path_effects: `.AbstractPathEffect`
  picker: float or callable[[Artist, Event], tuple[bool, dict]]
```

Advanced Matplotlib Concepts

getp / setp

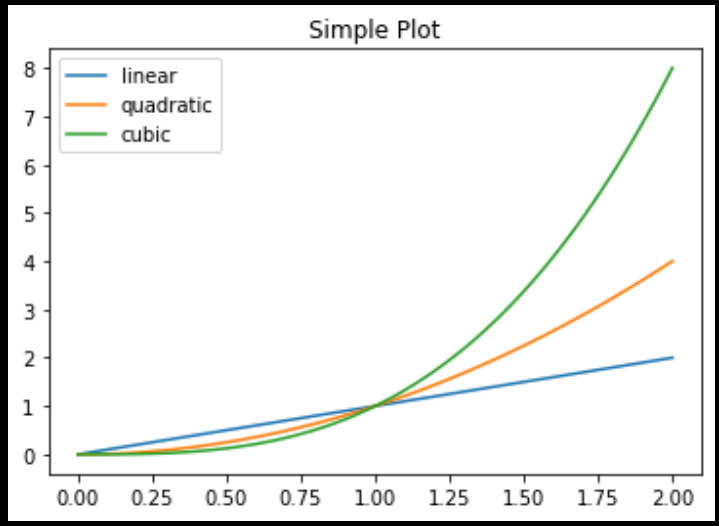
```
>>> plt.setp(l2, "linestyle")
  linestyle: {'-', '--', '-.', ':', '', (offset, on-off-seq), ...}

>>> plt.setp(l2, linestyle="-.", lw=5, color="red", alpha=0.5)
   [None, None, None, None]

>>> fig.get_figure()
```



Advanced Matplotlib Concepts

Legends

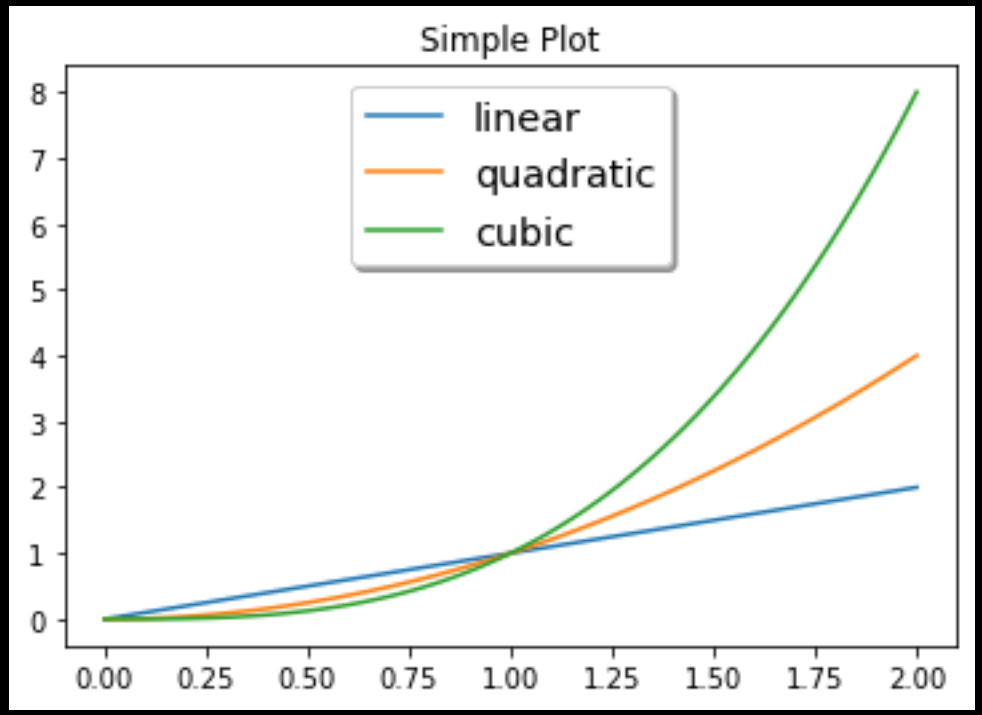
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)

fig, ax = plt.subplots()  # Create a figure and an axes.

l1 = ax.plot(x, x, label="linear")
l2 = ax.plot(x, x ** 2, label="quadratic")
l3 = ax.plot(x, x ** 3, label="cubic")

ax.set_title("Simple Plot")
ax.legend()
plt.show()
```



Advanced Matplotlib Concepts

Legends

```
ax.legend(loc='upper center', shadow=True, fontsize='x-large')
```



Advanced Matplotlib Concepts

Cycler

Matplotlib changes its colors or cycles through different styles all by itself. Matplotlib uses Python built-in objects called Cyclers.

```
from cycler import cycler

c1 = cycler(arg1=[1, 2, 3, 4])
c2 = cycler(arg2=list("rgba"))

for i in c2:
    print(i)

for i in c1 + c2:
    print(i)

{'arg2': 'r'}
{'arg2': 'g'}
{'arg2': 'b'}
{'arg2': 'a'}
{'arg1': 1, 'arg2': 'r'}
{'arg1': 2, 'arg2': 'g'}
{'arg1': 3, 'arg2': 'b'}
{'arg1': 4, 'arg2': 'a'}
```

Advanced Matplotlib Concepts

Cycler

Matplotlib changes its colors or cycles through different styles all by itself. Matplotlib uses Python built-in objects called Cyclers.

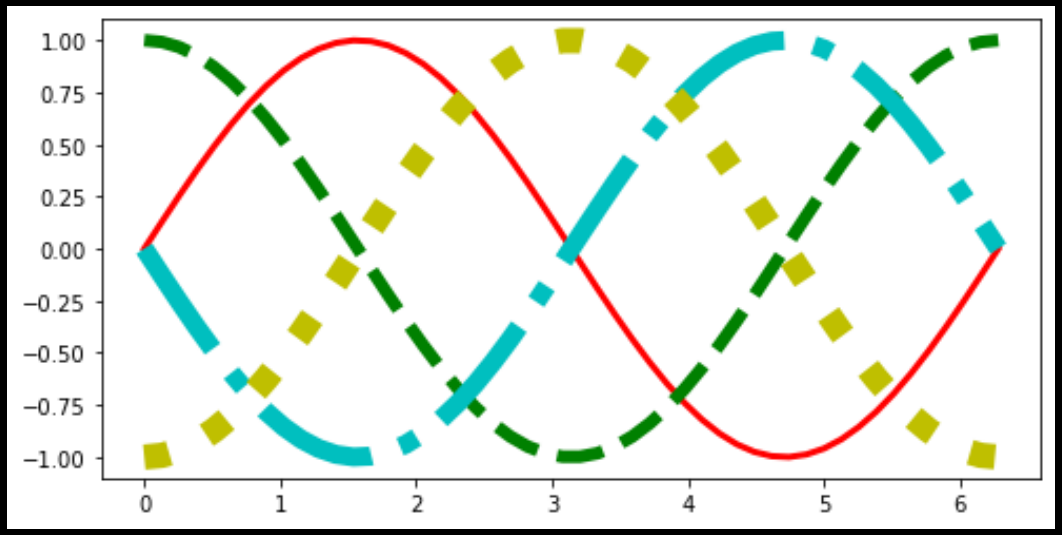
```
from cycler import cycler

line_prop_cycler = (
    cycler(color=list("rgcy"))
    + cycler(ls=["-", "--", "-.", ":"])
    + cycler(lw=[3, 6, 9, 12])
)

for i in line_prop_cycler:
    print(i)
```

```
{'color': 'r', 'ls': '-', 'lw': 3}
{'color': 'g', 'ls': '--', 'lw': 6}
{'color': 'c', 'ls': '-.', 'lw': 9}
{'color': 'y', 'ls': ':', 'lw': 12}
```

Advanced Matplotlib Concepts

Cycler

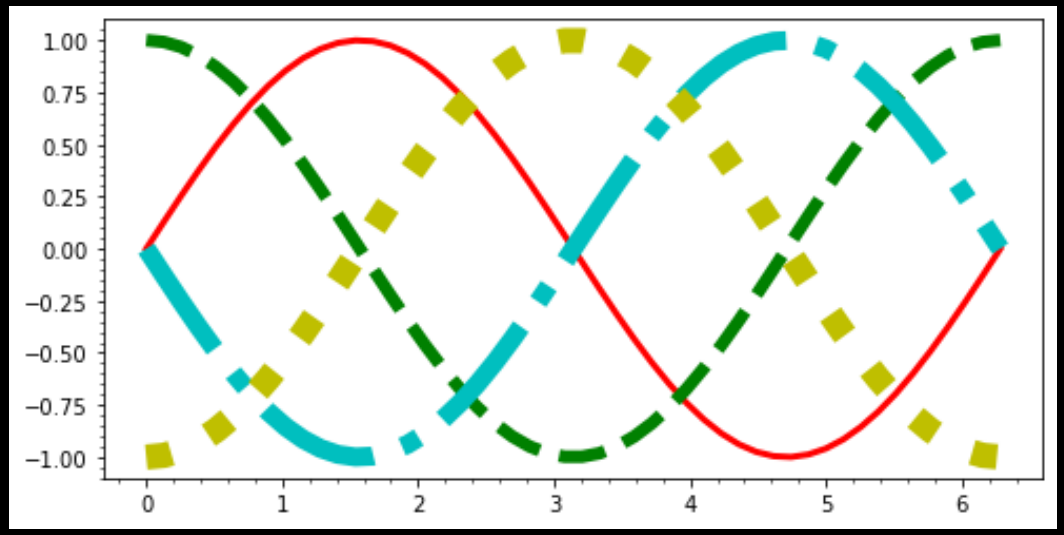
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 50)
offsets = np.linspace(0, 2 * np.pi, 4, endpoint=False)
yy = np.transpose([np.sin(x + phi) for phi in offsets])

fig, ax = plt.subplots(figsize=(8, 4))

ax.set_prop_cycle(line_prop_cycler)  # Set propcycle before plotting
ax.plot(x, yy)

plt.show();
```



Advanced Matplotlib Concepts

tick_params

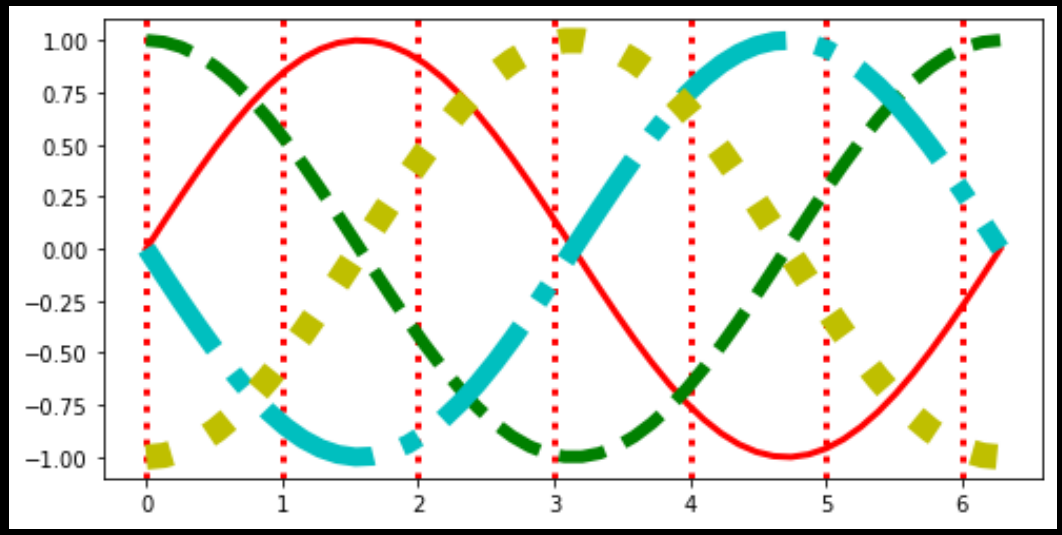
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 50)
offsets = np.linspace(0, 2 * np.pi, 4, endpoint=False)
yy = np.transpose([np.sin(x + phi) for phi in offsets])

fig, ax = plt.subplots(figsize=(8, 4))

ax.set_prop_cycle(line_prop_cycler)  # Set propcycle before plotting
ax.plot(x, yy)
ax.minorticks_on()
plt.show();
```



Advanced Matplotlib Concepts

Adding custom grids

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 50)
offsets = np.linspace(0, 2 * np.pi, 4, endpoint=False)
yy = np.transpose([np.sin(x + phi) for phi in offsets])
fig, ax = plt.subplots(figsize=(8, 4))
ax.set_prop_cycle(line_prop_cycler)  # Set propcycle before plotting
ax.plot(x, yy)
ax.grid(axis="x", linestyle=":", lw=3, color="r")
plt.show();
```

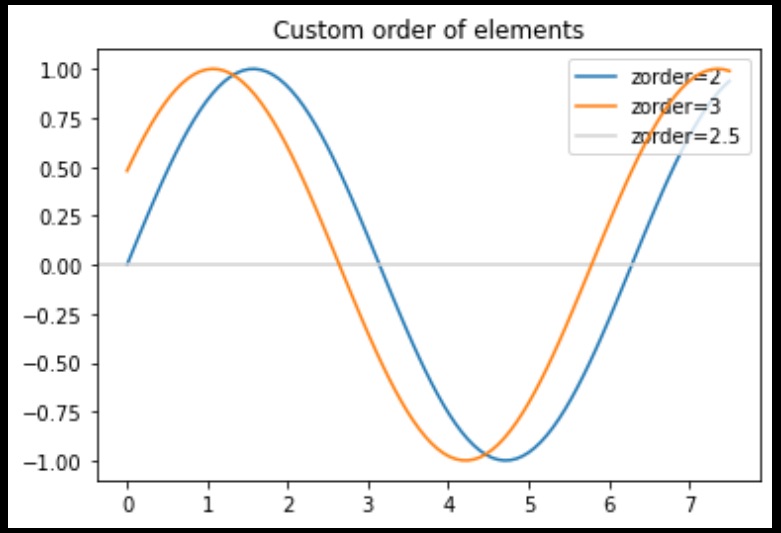


Advanced Matplotlib Concepts

Zorder

make sure that each plot gets drawn on the canvas in proper order

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 7.5, 100)
plt.plot(x, np.sin(x), label="zorder=2", zorder=2)  # bottom
plt.plot(x, np.sin(x + 0.5), label="zorder=3", zorder=3)
plt.axhline(0, label="zorder=2.5", color="lightgrey", zorder=2.5)
plt.title("Custom order of elements")
l = plt.legend(loc="upper right")
# legend between blue and orange line
l.set_zorder(2.5)
plt.show()
```



Displacement - Velocity

The **displacement** $\Delta x(t_1)$ over the time interval from $t = t_1$ to $t = t_1 + \Delta t$ is defined as:

$$\Delta x(t_1) = x(t_1 + \Delta t) - x(t_1)$$

The displacement is read directly from the motion diagram as the length of the line from *x(1 s)* to *x(2 s)*. The displacement has a direction—it is the displacement from *x(ti)* to $x(t_i + \Delta t)$.
The **average velocity** from $t = t_1$ to $t = t_1 + \Delta t$ is:

$$\hat{v}(t_i) = \frac{x(t_1 + \Delta t) - x(t_1)}{\Delta t} = \frac{\Delta x(t_1)}{\Delta t}$$

The average velocity has units meters per second, m/s.

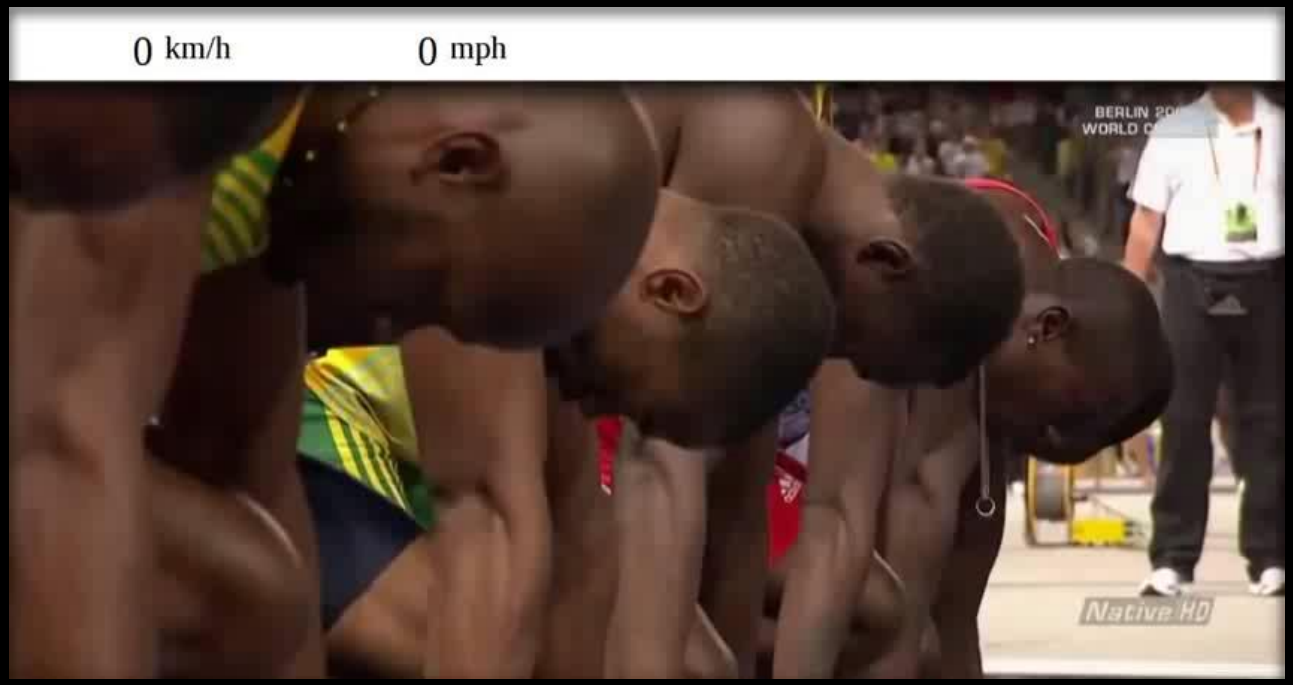
Velocity - Acceleration

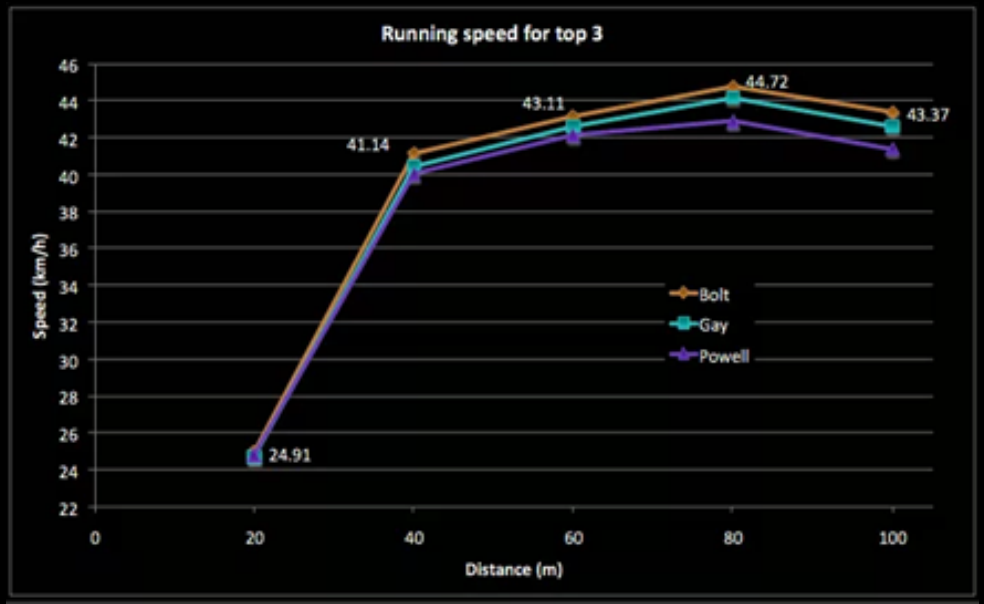
The velocity may also vary throughout the motion. As we introduced the velocity to characterize the rate of change of position, we introduce the acceleration to characterize the rate of change of the velocity:

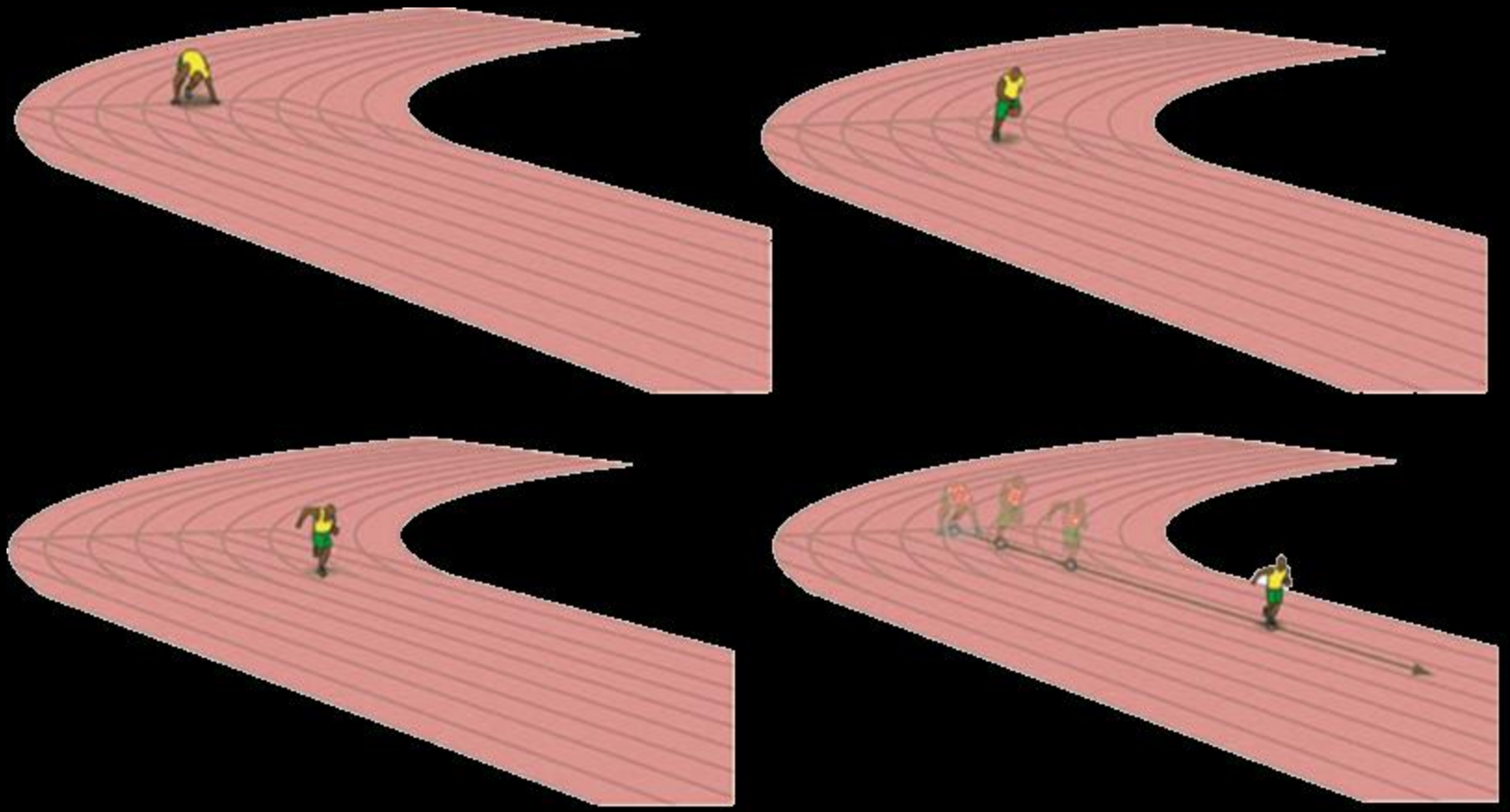
The **average acceleration** over a time interval Δt from t to t + Δt is:

$$\hat{a}(t_i) = \frac{v(t_1 + \Delta t) - v(t_1)}{\Delta t} = \frac{\Delta v(t_1)}{\Delta t}$$

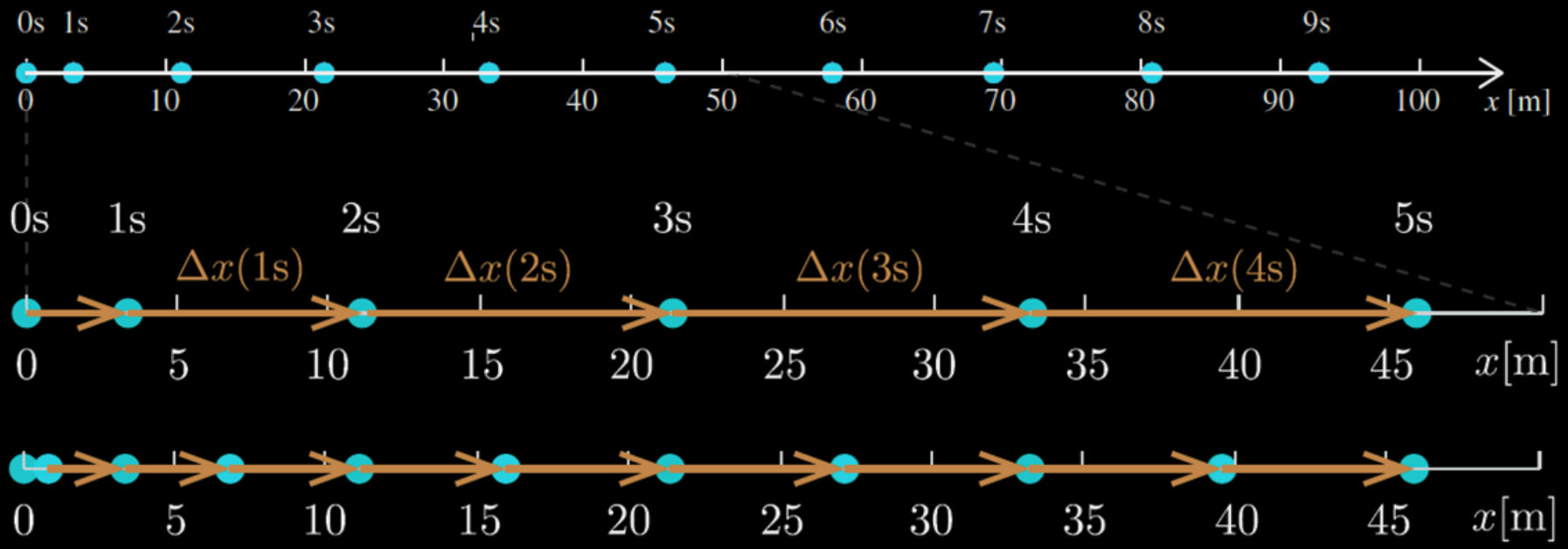
The average acceleration has units meters per second squared, m/s^2.


0 km/h
0 mph
Native HD

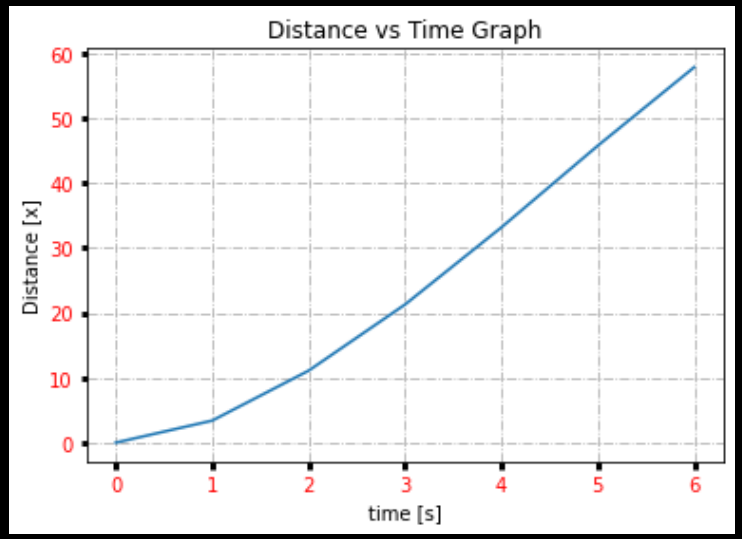

Running speed for top 3
Speed (km/h)
Distance (m)
46
44
42
40
38
36
34
32
30
28
26
24
22
0
20
40
60
80
100
41.14
43.11
44.72
43.37
24.91
Bolt
Gay
Powell



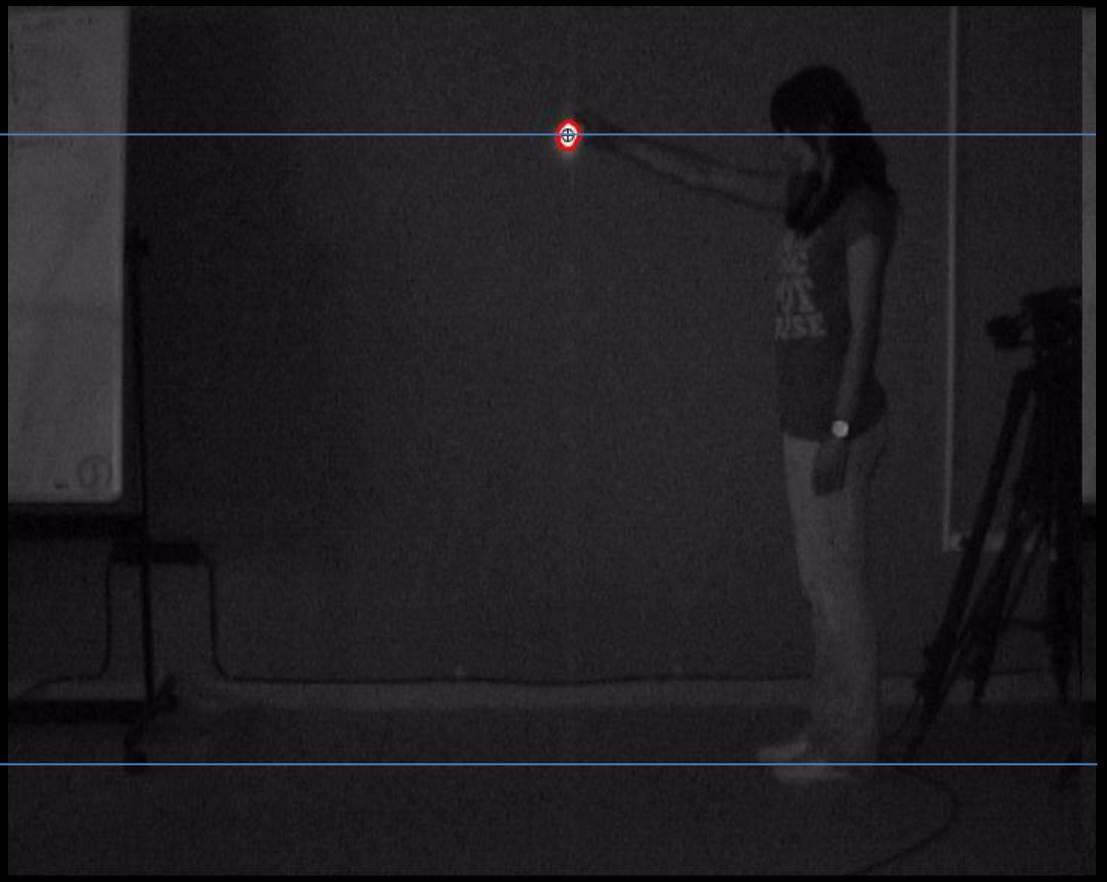
i	0	1	2	3	4	5	6
t_i (s)	0.0	1.0	2.0	3.0	4.0	5.0	6.0
x_i (m)	0.0	3.4	11.1	21.3	33.2	45.8	57.9




```
import matplotlib.pyplot as plt
import numpy as np
# Data for plotting
t = np.array([0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0])
x = np.array([0.0, 3.4, 11.1, 21.3, 33.2, 45.8, 57.9])

fig, ax = plt.subplots()
ax.plot(t, x)

ax.set(xlabel='time [s]',\
       ylabel='Distance [x]',
       title='Distance vs Time Graph')

ax.grid(True, linestyle='-.')
ax.tick_params(labelcolor='r', \
               labelsize='medium', \
               width=3)
plt.show()
```



Free Fall of an Object
How to calculate velocity and accelerations



X	Y
56.77	159.06
56.74	159.10
56.70	158.85
56.64	158.37
56.53	157.37
56.47	156.11
56.38	154.33
56.30	152.21
56.21	149.65
56.12	146.74
56.03	143.33
55.94	139.58
55.87	135.39
55.80	130.86
55.72	125.80
55.63	120.48
55.56	114.61
55.50	108.47
55.41	101.82
55.38	94.85
55.27	87.41
55.17	79.67
55.07	71.47
55.00	63.06
54.92	54.11
54.79	44.90
54.73	35.26
54.69	25.35
54.61	15.05
54.58	4.53

Free Fall of an Object
How to calculate velocity and accelerations

i	t_i (s)	y_i (m)	Δy_i (m)	$\bar{v}_i$ (m/s)	$\bar{a}_i$ (m/s^2)
1	0.0	1.60	−0.05	−0.5	
2	0.1	1.55	−0.15	−1.5	−10.0
3	0.2	1.40	−0.24	−2.4	−9.0
4	0.3	1.16	−0.34	−3.4	−10.0
5	0.4	0.82	−0.43	−4.3	−9.0
6	0.5	0.39			

Free Fall of an Object
How to calculate velocity and accelerations

The average velocities can be calculated from the data: For each ***i*** in Table we calculate the average velocity from t_i to t_{i+1} using:

$$v_i = \frac{y_{i+1} - y_i}{\Delta t}$$

The velocities are increasing in magnitude since the ball is accelerating downward. We estimate the average accelerations by

$$a_i = \frac{v_i - v_{i+1}}{\Delta t}$$

Free Fall of an Object
How to calculate velocity and accelerations

```
import matplotlib.pyplot as plt
import numpy as np

x, y = np.loadtxt('dropxy.txt', delimiter=',', usecols=[0,1],unpack=True )
n = len(x)
delta_t = 1/50       # 50 frames per second
time = np.linspace(0, (n-1)*delta_t, n)

vy = np.zeros(n-1, float)
for i in range(n-1):
    vy[i] = (y[i+1] - y[i])/delta_t

ay = np.zeros(n-2, float)
for i in range(n-2):
    ay[i] = (vy[i+1] - vy[i])/delta_t
```

Free Fall of an Object
How to calculate velocity and accelerations

```
fig, (ax1,ax2,ax3)= plt.subplots(3, 1)

ax1.plot(time, y, 'o-')
ax1.set_ylabel('y[cm]')

ax2.plot(time[0:n-1],vy, '.-')
ax2.set_ylabel('vy [cm/s]')

ax3.plot(time[1:n-1],ay, '.-')
ax3.set_ylabel('ay [cm/s^2]')
ax3.set_xlabel('time (s)')

plt.show()
```