Matplotlib Guide

Meher Krishna Patel

Created on : Octorber, 2017

Last updated : October, 2018

Table of contents

Chapter 1

Basic plots

1.1 Introduction

In this tutorial, Matplotlib library is discussed in detail, which is used for plotting the data. Our aim is to introduce the commonly used 'plot styles' and 'features' of the Matplotlib library, which are required for plotting the results obtained by the simulations or visualizing the data during machine learning process.

1.2 Data generation with Numpy

In this tutorial, Numpy library is used to generate the data for plotting. For this purpose, only 5 functions of numpy library are used, which are shown in Listing 1.1,

Listing 1.1: Data generation using Numpy

```
#numpyDataEx.py
# import numpy library with short name `np'
# outputs are shown as comments
import numpy as np

# 1. linspace(a, b, total_points)
x = np.linspace(2, 8, 4)
print(x) # [ 2.  4.  6.  8.]

# 2. sin(x)
sinx = np.sin(x) # x is consider as radian
print(sinx) # [ 0.90929743 -0.7568025  -0.2794155   0.98935825]

# 3. cos(x)
cosx = np.cos(x)
print(cosx) #[-0.41614684 -0.65364362  0.96017029 -0.14550003]

# 4. rand: uniform random variables
ur = np.random.rand(4) # result below will be different as it is random
print(ur) # [ 0.99791448  0.95621806  0.48124676  0.209090043]

# 5. randn: normal random variables
nr = np.random.randn(4) # result below will be different as it is random
print(nr) # [-0.37188868 -0.5680135  -0.21731407 -0.69523557]
```

1.3 Basic Plots

In this section, basic elements of the plot e.g. Title, axis names and grid etc. are discussed.

1.3.1 First Plot

Listing 1.2 plots the sin(x) as shown in Fig. Fig. 1.1.

- Explanation Listing 1.2
  Here, line 8 generates 100 equidistant points in the range $[-2\pi, 2\pi]$. Then line 9 calculates the sine values for those points. Line 10 plots the figure, which is displayed on the screen using line 11.

Listing 1.2: Sin(x), Fig. Fig. 1.1

```
1  #firstPlot.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # close all the figures, if open from previous commands
6  plt.close('all')
7
8  x=np.linspace(-2*np.pi, 2*np.pi, 100)
9  sinx=np.sin(x) # calculate sin(x)
10 plt.plot(x,sinx) #plot x on x-axis and sin_x on y-axis
11 plt.show() #display the plot
```

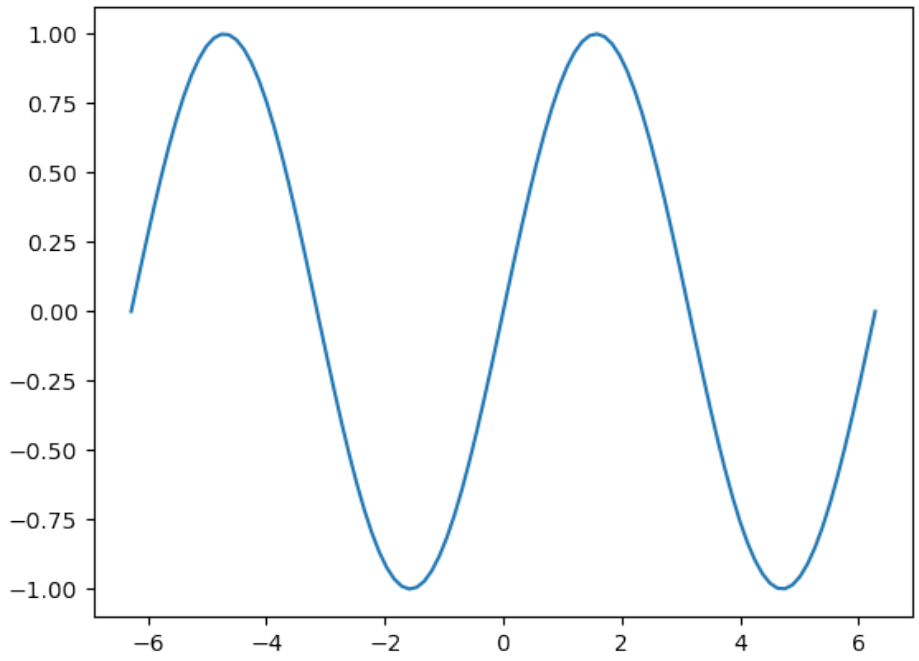

Fig. 1.1: fig_firstPlot

1.3.2 Label, Legend and Grid

Here Listing 1.2 is modified as shown in Listing 1.3, to add labels, legend and grid to the plot.

- Explanation Listing 1.3
  In line 11, label='sin' is added which is displayed by 'legend' command in line 15. 'loc=best' is optional parameter in line 15. This parameter find the best place for the legend i.e. the place where it does not touch the plotted curve. Line 18 and 19 add x and y label to curves. Finally, line 21 adds the grid-lines to the plot.
  For changing the location of legend, replace 'best' in line 15 with 'center', 'center left', 'center right', 'lower center', 'lower left', 'lower right', 'right', 'upper center', 'upper left' or 'upper right'.

Listing 1.3: Grid, Label and Legend, Fig. 1.2

```
1  #firstPlotGL.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # close all the figures, if open from previous commands
6  plt.close('all')
7
8  ############ Sin(x) ####################
9  x=np.linspace(-2*np.pi, 2*np.pi, 100)
10 sinx=np.sin(x) # calculate sin(x)
11 plt.plot(x,sinx, label='sin')
12
13 ############ Legend ####################
14 # label in plt.plot are displayed by legend command
15 plt.legend(loc="best") #show legend
16
17 #### Lable and Grid ####################
18 plt.xlabel("Radian") # x label
19 plt.ylabel("Amplitude") # y label
20
21 plt.grid() # show grid
22
23 plt.show() #display the plot
```

1.3.3 Line style and Marker

It is good to change the line styles and add markers to the plots with multiple graphs. It can be done as shown in Listing 1.4.

- Explanation Listing 1.4
  In line 13, '*–r' is the combination of three separate parameters i.e. '*', '–' and 'r', which represents 'marker', 'line style' and 'color' respectively. We can change the order of these combinations e.g. 'r–*' and '–r*' etc. Also, combination of two parameters (e.g. 'r–') or single parameter (e.g. '–') are valid.
  Table Table 1.1, Table 1.2 and Table 1.3 show some more abbreviations for 'line style', 'marker style' and 'color' respectively. Note that, only one element can be chosen from each style to make the combination.
  Further, line 13 contains 'markersize' parameter for changing the size of the marker. Table Table 1.4 shows the complete list of additional parameters to change the plot style. Lastly, line 13 can be rewritten using complete features of the plot as follows,

```
plt.plot(x, sinx, color='m',
    linestyle='-.', linewidth=4,
     marker='o', markerfacecolor='k', markeredgecolor='g',
     markeredgewidth=3, markersize=5,
    label='sin'
)
```

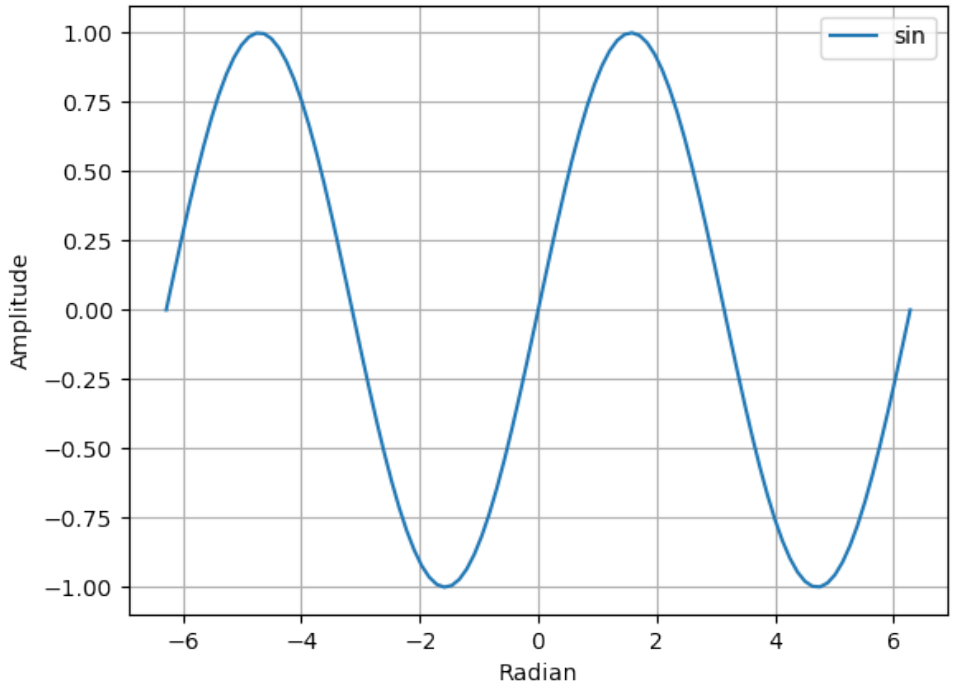

Fig. 1.2: Grid, Axes, Label and Legend, Listing 1.3

Listing 1.4: Line Style and Marker, Fig. 1.3

```
#firstPlotLineMarker.py
import numpy as np
import matplotlib.pyplot as plt

# close all the figures, if open from previous commands
plt.close('all')

############ Sin(x) ####################
x=np.linspace(-2*np.pi, 2*np.pi, 100)
sinx=np.sin(x) # calculate sin(x)

############ Line style and Marker ###################
plt.plot(x,sinx, '*--r', markersize=10, label='sin')

############ Legend ####################
plt.legend(loc="best") #show legend

#### Lable and Grid ####################
plt.xlabel("Radian") # x label
plt.ylabel("Amplitude") # y label
plt.grid() # show grid

plt.show() #display the plot
```

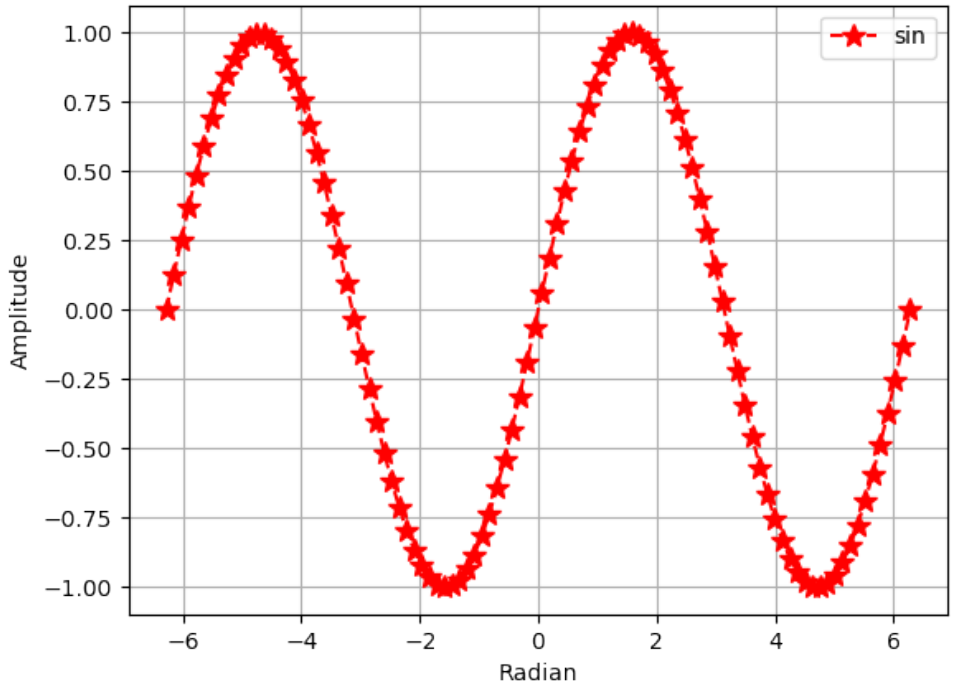

Fig. 1.3: Line-style and Line-marker, Listing 1.4

Table 1.1: Line styles

Keyword	Description
-.	dash-dot line
–	dashed line
:	dotted line
-	solid line

Table 1.2: Marker styles

Keyword	Description
o	circle
x	cross
D	diamond
h	hexagon
p	pentagon
+	plus
.	dot
s	square
*	star
V	down triangle
<	left triangle
>	right triangle
^	up triangle

Table 1.3: Color styles

Keyword	Description
k	black
b	blue
c	cyan
g	green
m	magenta
r	red
w	white
y	yellow

Table 1.4: Plot styles

color	set the color of line
linestyle	set the line style
linewidth	set the line width
marker	set the line-marker style
markeredgecolor	set the marker edge color
markeredgewidth	set the marker edge width
markerfacecolor	set the marker face color
markersize	set the marker size

1.3.4 Axis and Title

Listing 1.5 adds axis and title in previous figures.

- Explanation Listing 1.5

  Lines 25 and 26 in the listing add display range for x and y axis respectively in the plot as shown in Fig. 1.4. Line 30 and 32 add the ticks on these axis. Further, line 31 adds various 'display-name' for the ticks. It changes the display of ticks e.g. 'np.pi' is normally displayed as '3.14', but $r'+\backslash pi$ will display it as $+\pi$. Note that $\backslash pi$ is the 'latex notation' for π, and matplotlib supports latex notation as shown in various other examples as well in the tutorial.

Listing 1.5: Title and Axis, Fig. 1.4

```
1  #completeBasicEx.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # close all the figures, if open from previous commands
6  plt.close('all')
7
8  ############ Sin(x) ####################
9  x=np.linspace(-2*np.pi, 2*np.pi, 100)
10 sinx=np.sin(x) # calculate sin(x)
11
12 ############ Line style and Marker ##################
13 plt.plot(x,sinx, '*--r', markersize=10, label='sin')
14
15 ############ Legend ####################
16 plt.legend(loc="best") #show legend
17
18 #### Lable and Grid ####################
19 plt.xlabel("Radian") # x label
20 plt.ylabel("Amplitude") # y label
21 plt.grid() # show grid
```

(continues on next page)

(continued from previous page)

```
22
23  ############ Axis Limit and Marker ##################
24  # x-axis and y-axis limits
25  plt.xlim([-2*np.pi, 2*np.pi]) # x-axis display range
26  plt.ylim([-1.5, 1.5]) # y-axis display range
27
28  # ticks on the axis
29  # display ticks in pi format rather than 3.14 format
30  plt.xticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi],
31             [r'$-2\pi$', r'$-\pi$', r'$0$', r'$+\pi$', r'2$\pi$'])
32  plt.yticks([-1, 0, +1])
33
34  ################## Title ########################
35  plt.title("Plot $Sin(x)$")
36
37  plt.show()
```

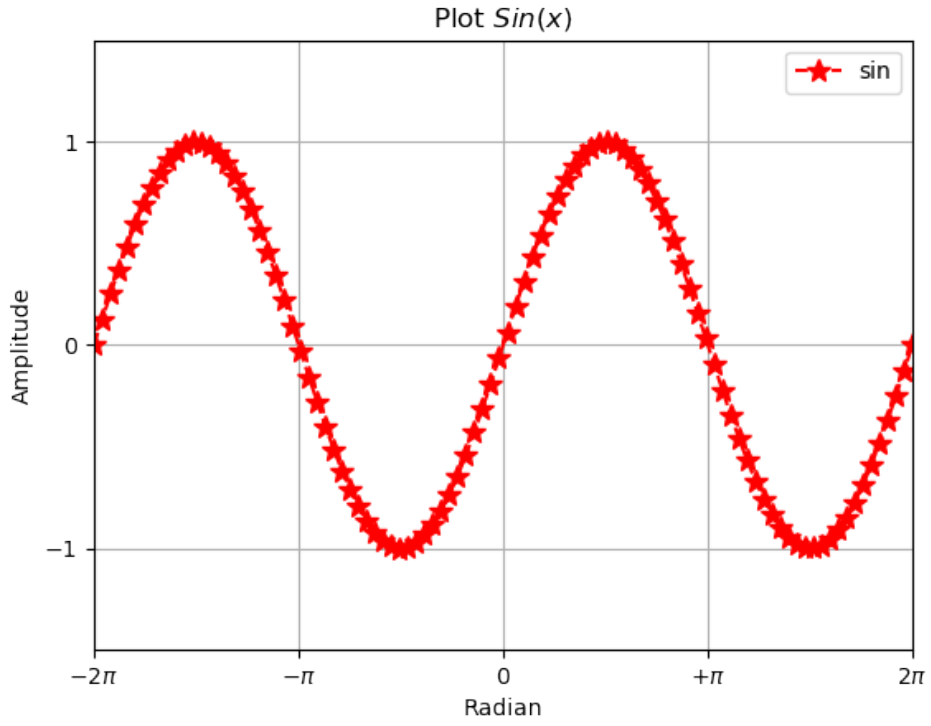

Fig. 1.4: Axis-limit and Axis-marker, Listing 1.5

1.4 Multiple Plots

In this section various mutliplots are discussed e.g. plots on the same figure window and subplots etc.

1.4.1 Mutliplots in same window

By default, matplotlib plots all the graphs in the same window by overlapping the figures. In Listing 1.6, line 14 and 15 generate two plots, which are displayed on the same figure window as shown in Fig. 1.5.

Listing 1.6: Mutliplots in same window, Fig. 1.5

```
#multiplot.py
import numpy as np
import matplotlib.pyplot as plt

# close all the figures, if open from previous commands
plt.close('all')

############ Sin(x) ####################
x=np.linspace(-2*np.pi, 2*np.pi, 100)
sinx=np.sin(x) # calculate sin(x)
cosx=np.cos(x) # calculate cos(x)

########### Line style and Marker ##################
plt.plot(x,sinx, '*--r', label='sin')
plt.plot(x,cosx, 'o-g', label='cos')

############ Legend ####################
plt.legend(loc="best") #show legend

#### Lable and Grid ####################
plt.xlabel(r'$Radian$').set_fontsize(16) # x label
plt.ylabel(r'$Amplitude$').set_fontsize(16) # y label
plt.grid() # show grid

plt.show()
```

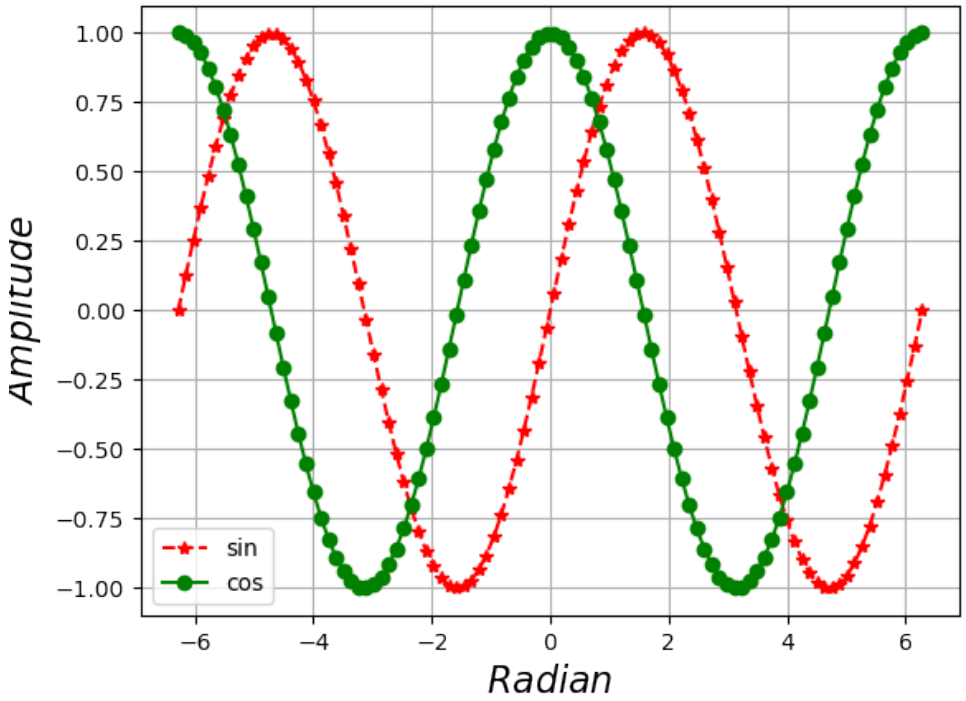

Fig. 1.5: Multiplots, Listing 1.6

1.4.2 Subplots

In Fig. 1.5, two plots are displayed in the same window. Further, we can divide the plot window in multiple sections for displaying each figure in different section as shown in Fig. 1.6

- Explanation Listing 1.7
  Subplot command takes three parameters i.e. number of rows, numbers of columns and location of the plot. For example in line 14, subplot(2,1,1), divides the figure in 2 rows and 1 column, and uses location 1 (i.e. top) to display the figure. Similarly, in line 21, subplot(2,1,2) uses the location 2 (i.e. bottom) to plot the graph. Further, Listing 2.2 divides the figure window in 4 parts and then location 1 (top-left), 2 (top-right), 3 (bottom-left) and 4 (bottom-right) are used to display the graphs.
  Also, all the plot commands between line 14 and 21, e.g. line 15, will be displayed on the top location. Further, plots defined below line 21 will be displayed by bottom plot window.

Listing 1.7: Subplots, Fig. 1.6

```
1  #subplotEx.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # close all the figures, if open from previous commands
6  plt.close('all')
7
8  ############ Sin(x) ####################
9  x=np.linspace(-2*np.pi, 2*np.pi, 100)
10 sinx=np.sin(x) # calculate sin(x)
11 cosx=np.cos(x) # calculate cos(x)
12
13 ########### Subplot ###################
14 plt.subplot(2,1,1)
15 plt.plot(x,sinx, '*--r', label='sin')
16 plt.grid() # show grid
17 plt.legend() #show legend
18 plt.xlabel(r'$Radian$')# x label
19 plt.ylabel(r'$Amplitude$') # y label
20
21 plt.subplot(2,1,2)
22 plt.plot(x,cosx, 'o-g', label='cos')
23 plt.grid() # show grid
24 plt.legend() #show legend
25 plt.xlabel(r'$Radian$') # x label
26 plt.ylabel(r'$Amplitude$') # y label
27 ############ Legend ####################
28
29 #### Lable and Grid ####################
30 plt.xlabel(r'$Radian$') # x label
31 plt.ylabel(r'$Amplitude$') # y label
32
33
34 plt.show()
```

1.4.3 Mutliplots in different windows

'figure()' command is used to plot the graphs in different windows, as shown in line 17 of Listing 1.8.

- Explanation Listing 1.8
  Here optional name 'Sin' is given to the plot which is displayed on the top in Fig. 1.7. Then line 19 opens a new plot window with name 'Cos' as shown in Fig. 1.8. Finally in line 21, the name 'Sin' is used again, which selects the previously open 'Sin' plot window and plot the figure there. Hence, we get two plots in this window (i.e. from lines 18 and 22) as show in Fig. 1.7.

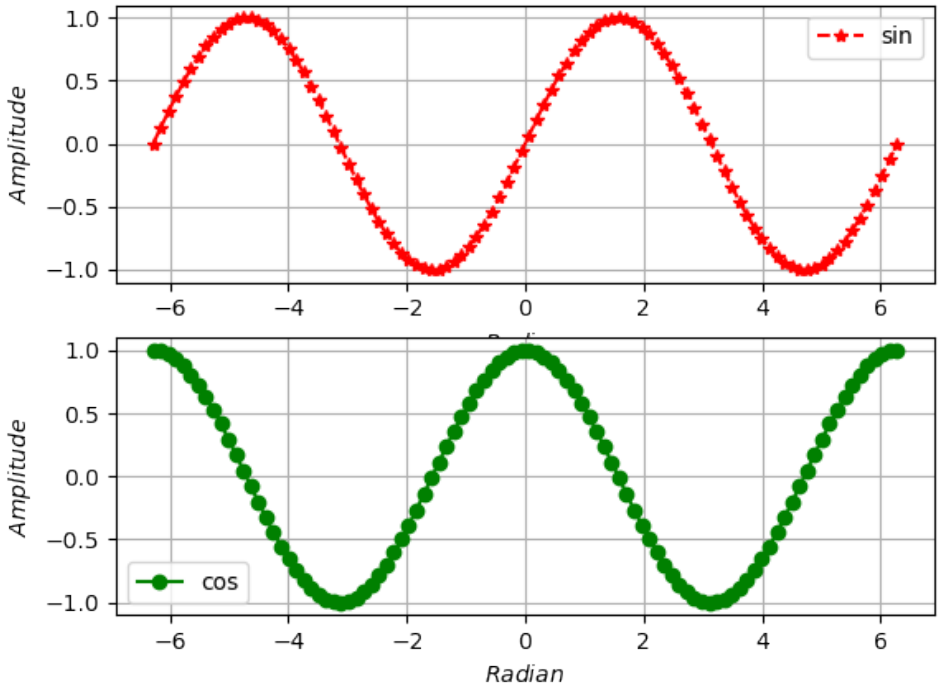

Fig. 1.6: Subplots, Listing 1.7

Listing 1.8: Mutliplots in different windows, Fig. 1.7 and Fig. 1.8

```
#multiplotDifferentWindow.py
import numpy as np
import matplotlib.pyplot as plt

# close all the figures, if open from previous commands
plt.close('all')

############ Sin(x) ####################
x=np.linspace(-2*np.pi, 2*np.pi, 100)
sinx=np.sin(x) # calculate sin(x)
cosx=np.cos(x) # calculate cos(x)

########### Open figure in seperate window ###################
#"Sin" is the name of figure window
# if not give i.e. plt.figure(), then 1 will be assigned to figure
# and for second plt.figure(), 2 will be assigned...
plt.figure("Sin")
plt.plot(x,sinx, '*--r', label='sin')
plt.figure("Cos")
plt.plot(x,cosx, 'o-g', label='cos')
plt.figure("Sin")
plt.plot(x,cosx, 'o-g', label='cos')

############ Legend ####################
plt.legend(loc="best") #show legend

#### Lable and Grid ####################
plt.xlabel(r'$Radian$').set_fontsize(16) # x label
plt.ylabel(r'$Amplitude$').set_fontsize(16) # y label
```

(continues on next page)

(continued from previous page)

```
30 plt.grid() # show grid
31
32 plt.show()
```

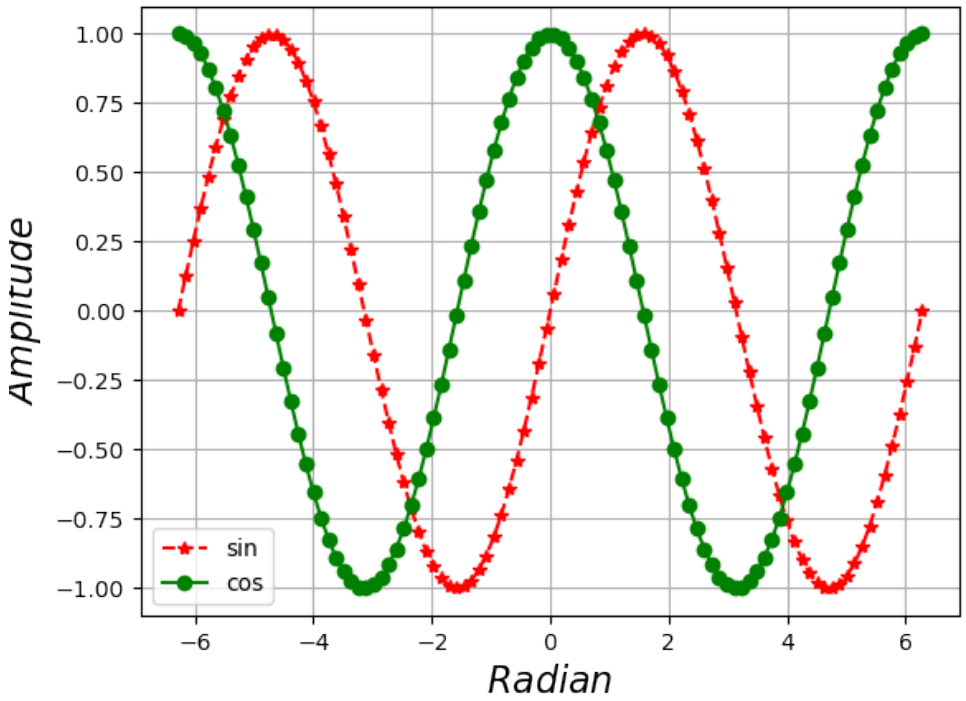

Fig. 1.7: Figure window with name 'Sin'

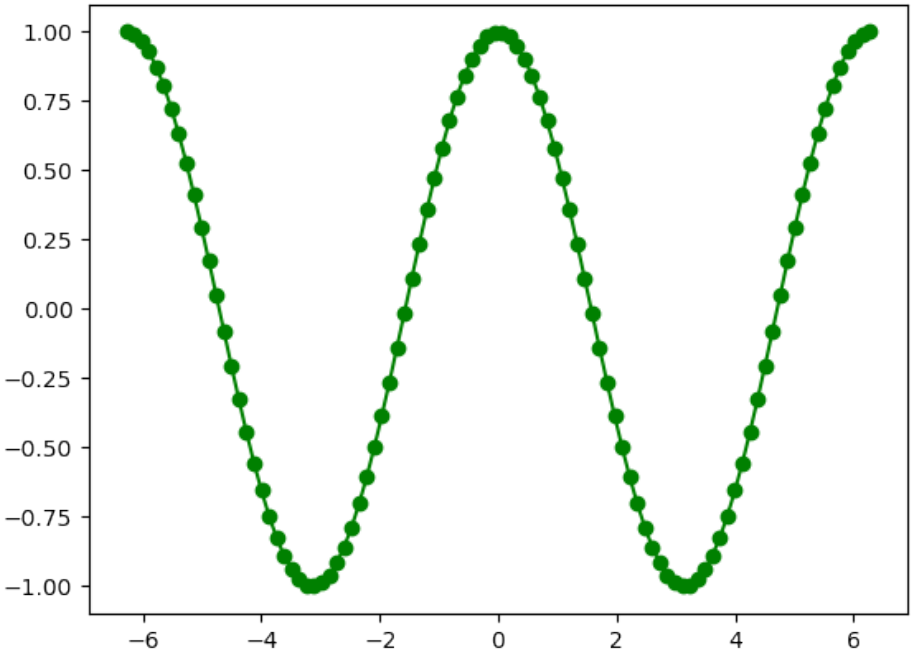

Fig. 1.8: Figure window with name 'Cos'

Chapter 2

Plot types

In this chapter, various plot types are discussed.

2.1 Semilog Plot

Semilog plots are the plots which have y-axis as log-scale and x-axis as linear scale as shown in Fig. 2.2. Listing 2.1 plots both the semilog and linear plot of the function e^x.

Listing 2.1: Linear plot (Fig. 2.1) vs Semilog plot (Fig. 2.2)

```
1  #semilogEx.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # close all the figures, if open from previous commands
6  plt.close('all')
7
8  x=np.linspace(0.01, 5, 100)
9  e=np.exp(x)
10
11 #linear plot
12 plt.plot(x,e)
13 plt.xlabel("x")
14 plt.ylabel("y=exp(x)")
15 plt.title("Linear Y axis")
16
17 #semilog plot
18 #log(exp(x))=x therefore straight line will be displayed
19 plt.figure()
20 plt.semilogy(x,e) #semilogy: semilog y-axis
21 plt.xlabel("x")
22 plt.ylabel("y=exp(x)")
23 plt.title("Log Y axis")
24
25 plt.show() #display the plot
```

2.2 Histogram

Histogram can be generated using hist() command as illustrated in line 11 in Listing 2.2. By default it generates 10 bins, which can be increased by providing the number of bins as shown in line 15. Further from Fig. 2.3, we can

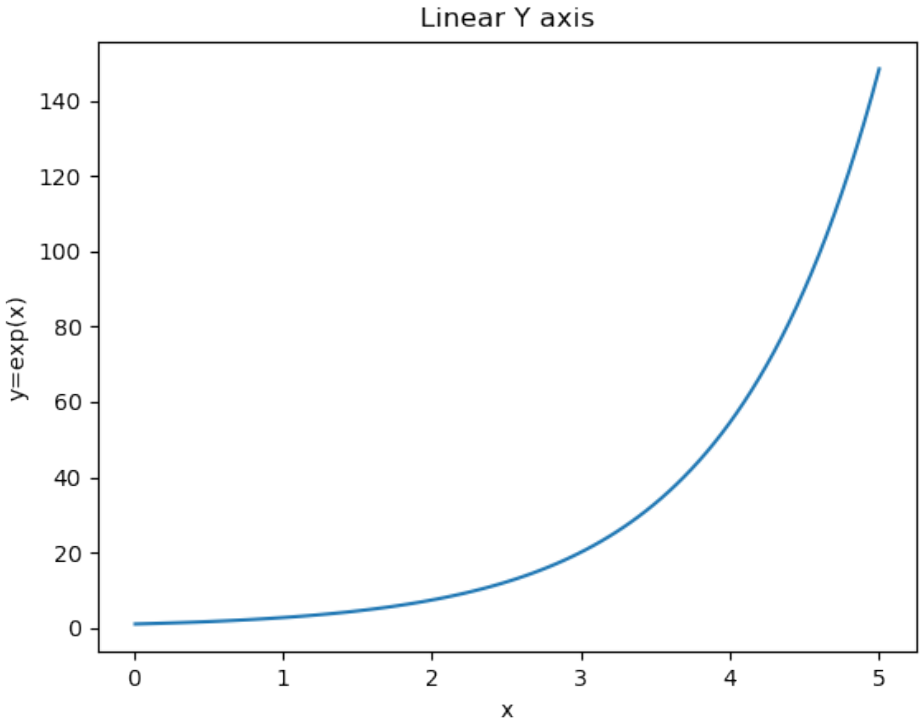

Fig. 2.1: Linear Plot

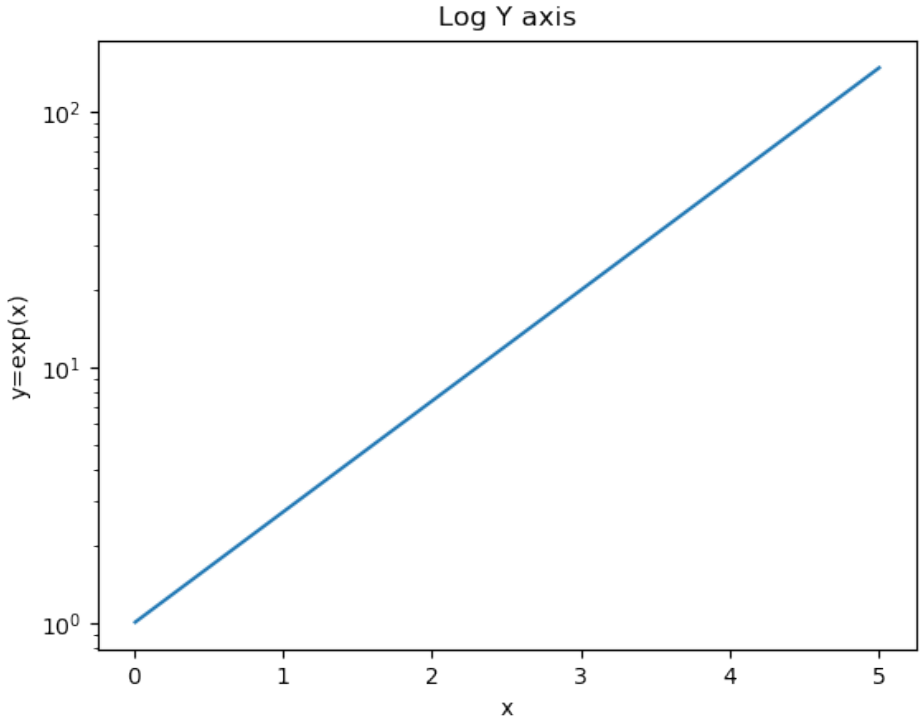

Fig. 2.2: Semilog Plot

see that 'rand' generates the random number in the range [0,1] with uniform density, whereas 'randn' generates the random number in the range [-1,1] with Gaussian (Normal) density.

Listing 2.2: Histogram, Fig. 2.3

```
#histogramEx.py
import numpy as np
import matplotlib.pyplot as plt

plt.close("all")

ur = np.random.rand(10000)
nr = np.random.randn(10000)

plt.subplot(2,2,1)
plt.hist(ur)
plt.xlabel("Uniform Random Number, Default 10 Bin")

plt.subplot(2,2,2)
plt.hist(ur, 20) # display 20 bins
plt.xlabel("Uniform Random Number, 20 Bin")

plt.subplot(2,2,3)
plt.hist(nr)
plt.xlabel("Normal Random Number, Default 10 Bin")

plt.subplot(2,2,4)
plt.hist(nr, 20) # display 20 bins
plt.xlabel("Normal Random Number, 20 Bin")

plt.show()
```

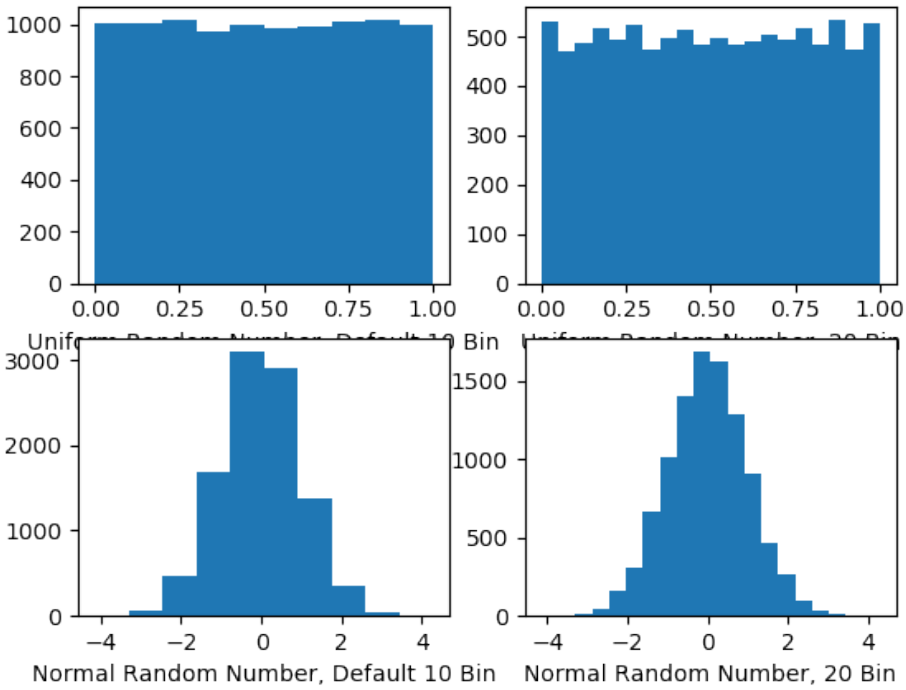

Fig. 2.3: Histograms, Listing 2.2

2.3 Scatter plot

Scatter plots are similar to simple plots and often use to show the correlation between two variables. Listing 2.3 generates two scatter plots (line 14 and 19) for different noise conditions, as shown in Fig. 2.4. Here, the distortion in the sine wave with increase in the noise level, is illustrated with the help of scatter plot.

Listing 2.3: Scatter plot, Fig. 2.4

```
1  #scatterEx.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  plt.close("all")
6
7  N=100
8  x = np.linspace(0, 2*np.pi, N)
9  noise = np.random.randn(N)
10 signal = 2*np.sin(x)
11
12 y = signal + noise
13 plt.plot(x, signal) # signal + noise
14 plt.scatter(x, y) #scatter plot
15
16 plt.figure()
17 y = signal + 0.2*noise # singal + 0.2*noise i.e. low noise
18 plt.plot(x, signal)
19 plt.scatter(x, y) #scatter plot
20
21 plt.show()
```

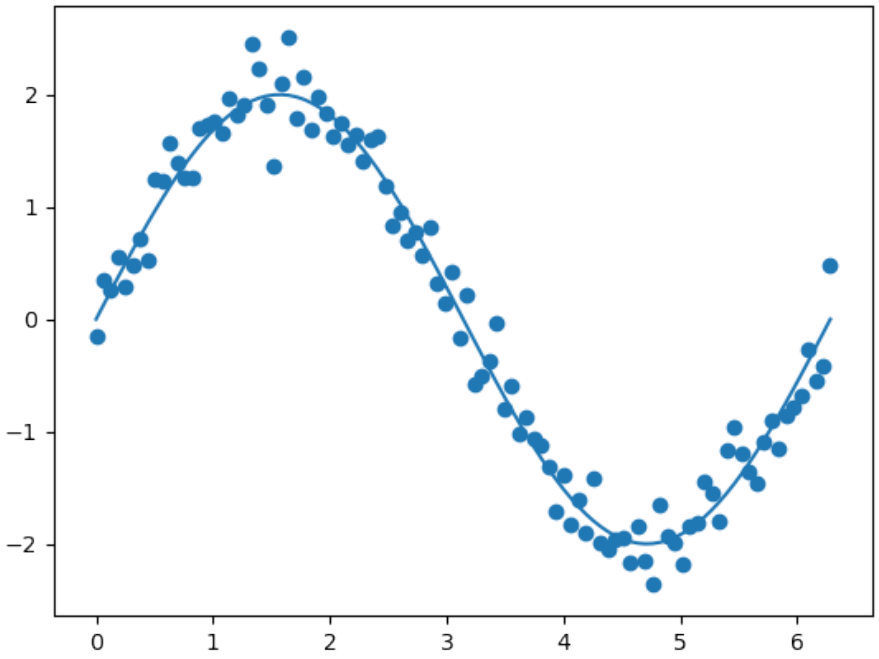

Fig. 2.4: Scatter Plot, Listing 2.3

2.4 Pie chart

Here, pie charts are generated in two formats. Listing 2.4 generates a simple Pie chart with data names as show in Fig. 2.5. Also in line 9, figsize=(5,5) command is used here, to resize the output figure window. Further, Listing 2.5 adds additional features (line 13) to it i.e. explode and auto-percentage as shown in Fig. 2.6.

Listing 2.4: Pie chart, Fig. 2.5

```
1  #pieEx.py
2  import matplotlib.pyplot as plt
3
4  plt.close("all")
5
6  x = [30, 20, 15, 25, 10]
7  dataName = ['data1', 'data2', 'data3', 'data4', 'data5']
8
9  plt.figure(figsize=(5,5)) #figsize: output figure window size
10 plt.pie(x, labels=dataName)
11
12 plt.show()
```

Listing 2.5: Pie chart: explode and auto-percentage, Fig. 2.6

```
1  #pieEx2.py
2  import matplotlib.pyplot as plt
3
4  plt.close("all")
5
6  x = [10, 10, 20, 20, 30]
7  dataName = ['data1', 'data2', 'data3', 'data4', 'data5']
8  explode = [0.01, 0, 0, 0.04, 0.09]
9
10 plt.figure(figsize=(5,5))
11 # autopct='%.2f %%': %.2f display value upto 2 decimal,
12 # %% is used for displaying % at the end
13 plt.pie(x, explode=explode, labels=dataName, autopct='%.2f%%')
14 plt.show()
```

2.5 Polar plot

In matplotlib, polar plots are based on clipping of the curve so that $rge0$. For example, in Fig. 2.7 (generated by line 12 in Listing 2.6), only two lobes of $cos(2x)$ are generated instead of four. Other two lobes have negative value of 'r', therefore these are clipped by the matplotlib. The actual four lobes are shown in Fig. 2.8, which can not be generated by matplotlib.

Listing 2.6: Polar plot, Fig. 2.7

```
1  #polarplotEx.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  plt.close("all")
6
7  x=np.linspace(0,2*np.pi, 1000)
8
9  # polar axes is based on clipping so that r >= 0.
10 # therefore only 2 lobes are shown as oppose to 4 lobes.
11 y = np.cos(2*x)
```

(continues on next page)

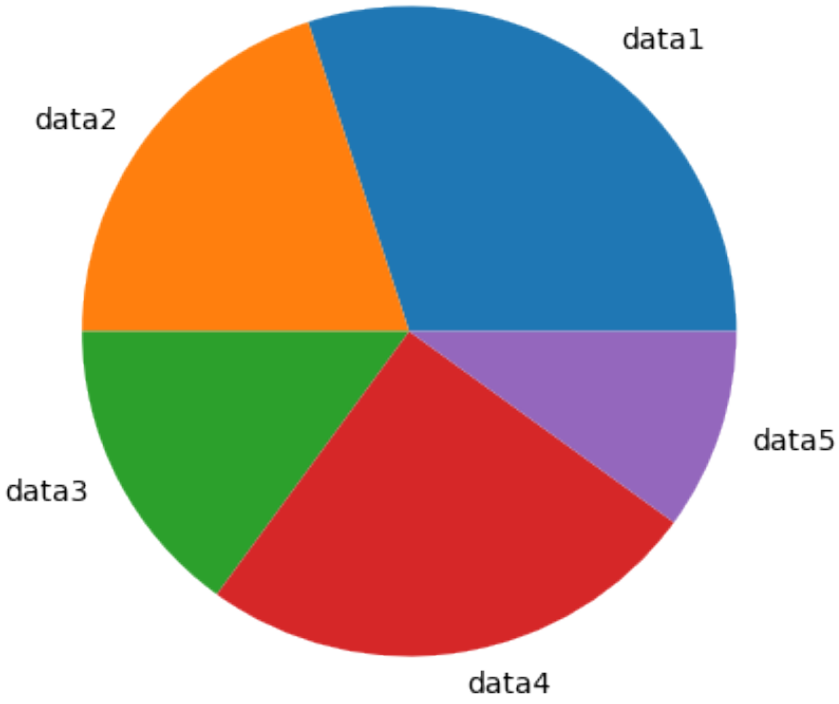

Fig. 2.5: Pie Chart with labels, Listing 2.4

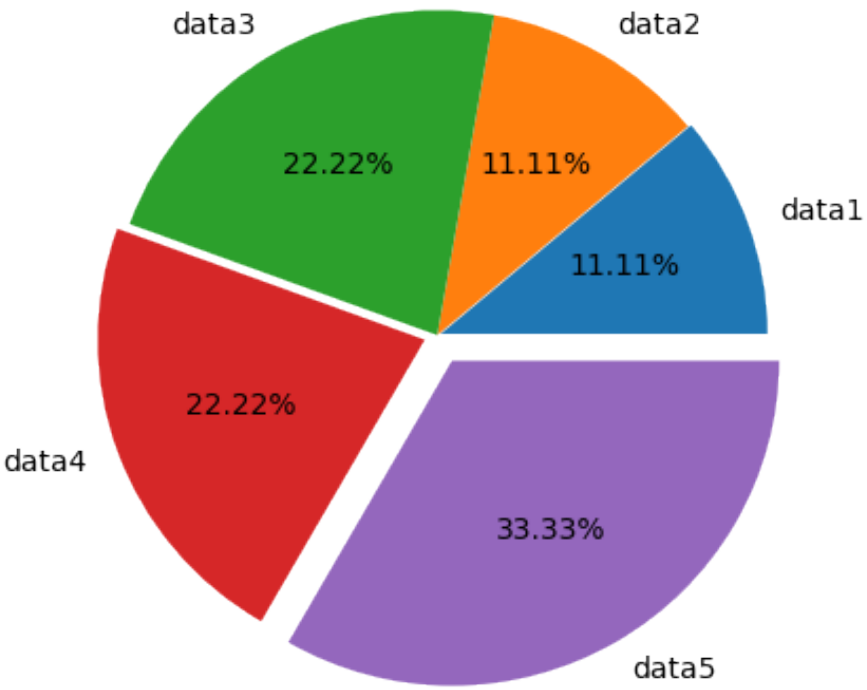

Fig. 2.6: Explode and Auto Percentage, Listing 2.5

(continued from previous page)

```
12 plt.polar(x, y)
13
14 plt.show()
```

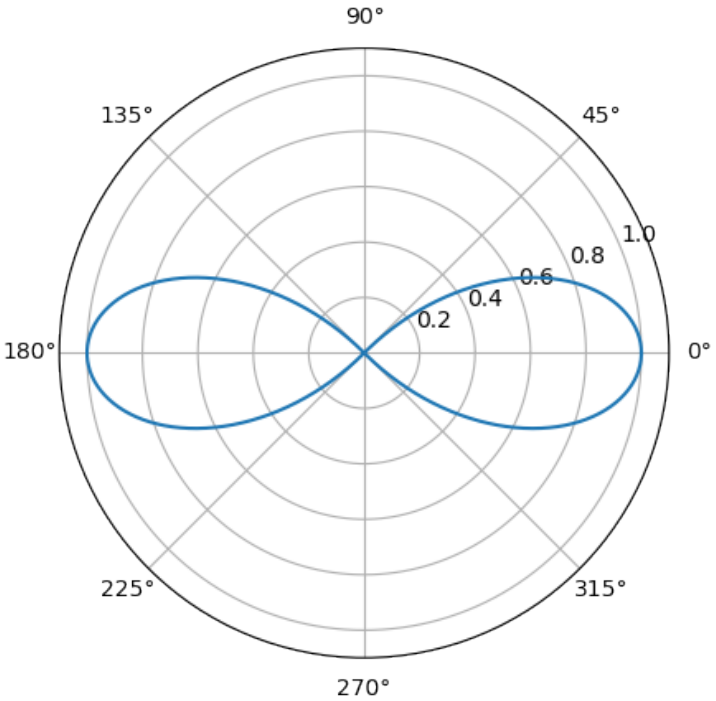

Fig. 2.7: Matplotlib: polar plot of Cos(2x)

2.6 Bar chart

In this section, two types of bar charts are discussed. Listing 2.7 plots a simple bar chart for 'years vs x'; whereas Listing 2.8 plots multiple data.

Listing 2.7: Bar Chart, Fig. 2.9

```
1  #barchartEx.py
2  import matplotlib.pyplot as plt
3
4  plt.close("all")
5
6  x = [1, 2, 4]
7  years = [1999, 2014, 2030]
8
9  plt.bar(years, x)
10
11 plt.show()
```

- Explanation Listing 2.8

  In Fig. 2.10, the data 'increment' is plotted above the data 'A' using 'bottom' parameter in line 14.

  Further, in the lower subplot, bar charts are plotted side by side using combination of 'locs' and 'width' variable in line 24 and 25. 'width' parameter set the width of the bar; which is set to 0.2

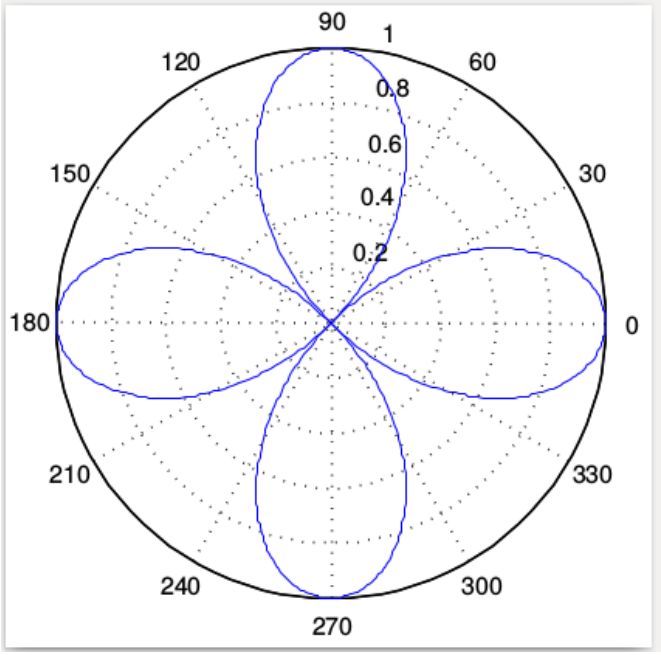

Fig. 2.8: Actual: polar plot of Cos(2x)

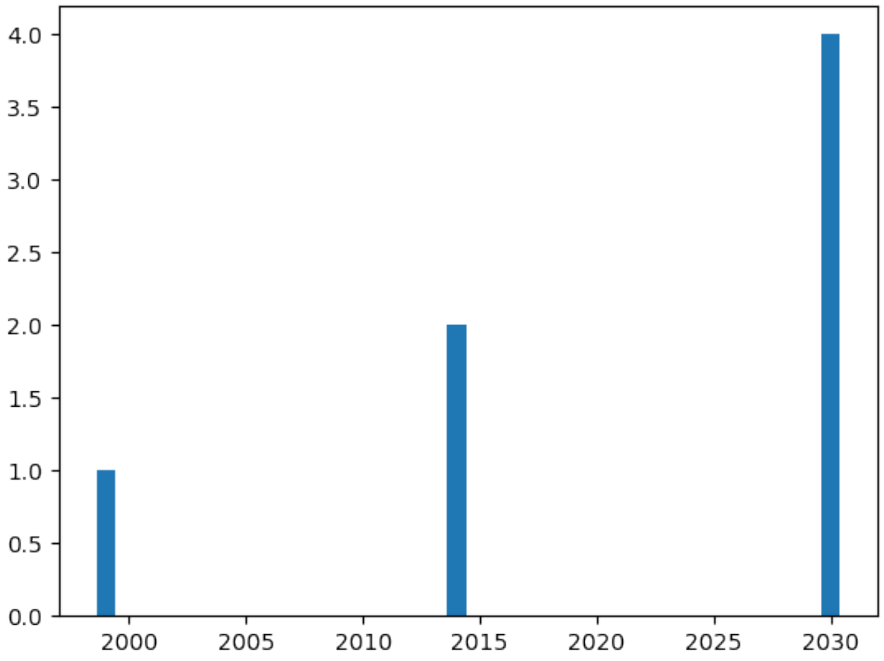

Fig. 2.9: Bar Chart with single data, Listing 2.7

in line 24. Line 27 plots the data 'x' first; then, line 28 plots next data set i.e. 'y', but location is shifted by the 'width' due to command 'locs+width' in the line. Hence, bar chart is plotted beside the bars of the line 27. After that, line 29 shifted the plot of data 'z' by '2*width'. Finally line 32 add ticks for the x-axis and we get the final plot as shown in Fig. 2.10.

Listing 2.8: Bar Chart with multiple data, Fig. 2.10

```
1  #barchartCombineEx.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  plt.close("all")
6
7  #### subplot 1
8  A = [2, 4, 8, 6]
9  increment = [5, 6, 4, 1]
10 years = [2005, 2010, 2015, 2020]
11
12 plt.subplot(2,1,1)
13 plt.bar(years, A, color = 'b', label='A')
14 plt.bar(years, increment, color = 'r', bottom = A, label='increment')
15 plt.legend()
16
17 plt.show()
18
19 ##### subplot 2
20 x = [1, 2, 4]
21 y = [3.5, 3, 2]
22 z = [2, 3, 1.5]
23
24 width = 0.2
25 locs = np.arange(1, len(x)+1)
26 plt.subplot(2,1,2)
27 plt.bar(locs, x, width=width, label='x')
28 plt.bar(locs+width, y, width=width, color="red", label='y')
29 plt.bar(locs+2*width, z, width=width, color="black", label='z')
30 plt.legend()
31
32 plt.xticks([1.25, 2.25, 3.25],
33           [r'$2000$', r'$2005$', r'$2010$'])
34
35 plt.show()
```

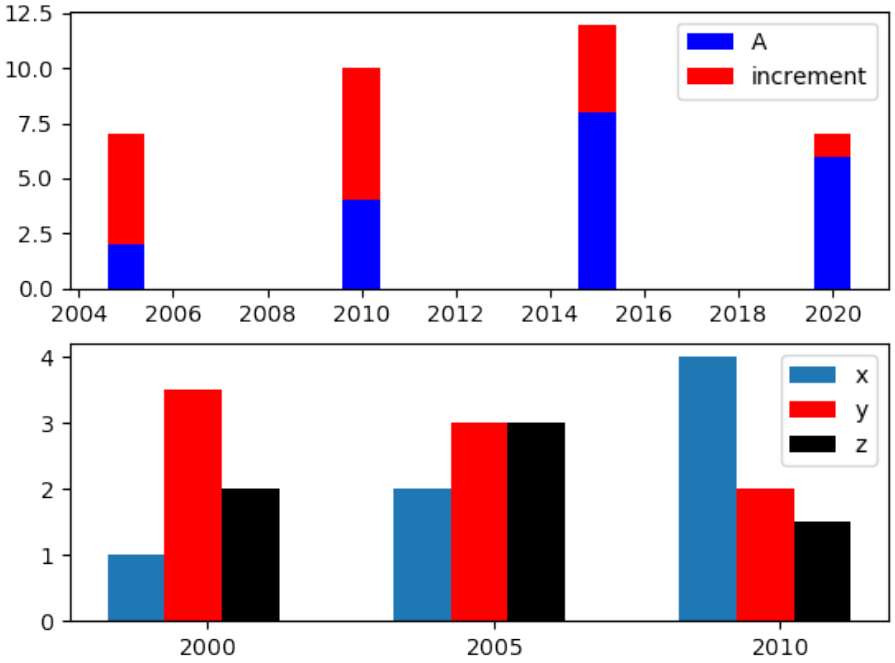

Fig. 2.10: Bar chart with multiple data, Listing 2.8

Chapter 3

Miscellaneous

3.1 Annotation

Annotation can be used to make graph more readable as show in Fig. 3.1. Text is added to graph using 'annotate()' command with two different methods as shown in line 25 and 40 in Listing 3.1. Also, 'text()' command (at line 49) is used to add text to the figure.

Listing 3.1: Annotation, Fig. 3.1

```
1  #annotateEx.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # close all the figures, if open from previous commands
6  plt.close('all')
7
8  ############ Sin(x) ####################
9  x=np.linspace(0, 2*np.pi, 100)
10 sinx=np.sin(x) # calculate sin(x)
11 plt.plot(x,sinx, label='sin') # legend
12
13 ############ Legend ####################
14 plt.legend(loc="best") #show legend
15
16 #### Lable and Grid ####################
17 plt.xlabel("Radian") # x label
18 plt.ylabel("Amplitude") # y label
19
20 ############# Annotate ######################
21 #1.
22 # other arrow style
23 # '-', '->', '-[', '<-', '<->', 'fancy', 'simple', 'wedge'
24 #sin(pi/2)=1
25 plt.annotate(r'$sin(\frac{\pi}{2})=1$',
26              fontsize=16,
27              xy=(1.5, 1), xytext=(1 , 0.5),
28              arrowprops=dict(arrowstyle="->"),
29         )
30
31 # 2.
32 #============================================================
33 # width: The width of the arrow in points
34 # frac: The fraction of the arrow length occupied by the head
35 # headwidth: The width of the base of the arrow head in points
```

(continues on next page)

(continued from previous page)

```
# shrink: Moves the tip and the base of the arrow some percent
# away from the annotated point and text,
#============================================================
#sin(3*pi/2)=-1
plt.annotate(r'$sin(\frac{3\pi}{2})=-1$',
             fontsize=18,
             xy=(4.5, -1), xytext=(1.5 , -0.5),
             arrowprops=dict(facecolor='red', shrink = 0.04,
                             connectionstyle="arc3,rad=.2"),
        )

# 3.
#################### Add text to plot ###########
plt.text(5, 0.5, 'This is \nSine wave');

plt.show() #display the plot
```

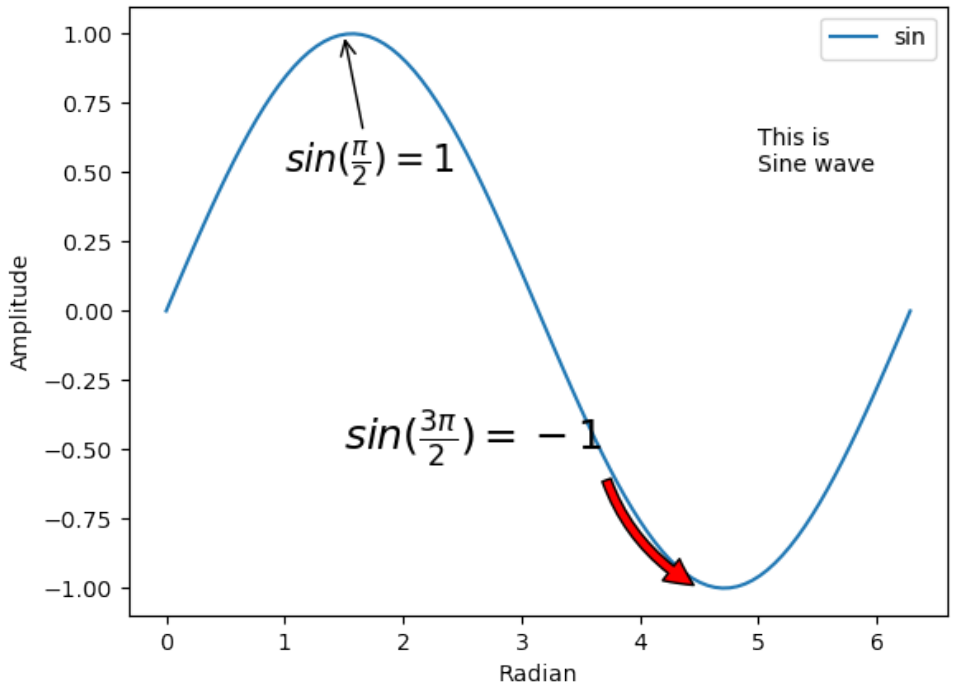

Fig. 3.1: Annotation, Listing 3.1

3.2 Sharing Axis

Listing Listing 3.2, Listing 3.3 and Listing 3.4 create the instances of figure() and subplot() functions of matplotlib to generate various plots.

3.2.1 Common axis for two plots

Here x axis is common for two different plots. Further, in one plot log y-axis is used.

- Explanation Listing 3.2

Line 11 creates an instance 'fig1' of figure() function. Then subfig1 and subfig2 instances of 'fig1' are created in line 14 and 15. 'twinx()' command in line 18, shares the x-axis for both the plots. Line 22-24 set the various parameter for subfig1; also note that 'set_' is used for x and y labels. Then line 27-28 plots the second figure. Finally line 30 displays both the plots as shown in Fig. 3.2

Listing 3.2: Sharing Axis, Fig. 3.2

```
1  #shareaxisEx.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  N=100
6  x=np.linspace(0.1,4, N)
7  e1 = np.exp(x)
8  e2 = np.exp(-x)
9
10 # create an object 'fig1'  of figure()
11 fig1 = plt.figure()
12
13 # create two instances of fig1 at location 1,1,1
14 subfig1 = fig1.add_subplot(1,1,1)
15 subfig2 = fig1.add_subplot(1,1,1)
16
17 # share the x-axis for both plot
18 subfig2 = subfig1.twinx()
19
20 # plot subfig1
21 # semilogy for log 'y' axis, for log x use semilogx
22 subfig1.semilogy(x, e1)
23 subfig1.set_ylabel("log scale: exp(x)")
24 subfig1.set_xlabel("x-->")
25
26 # plot subfig2
27 subfig2.plot(x, e2, 'r')
28 subfig2.set_ylabel("simple scale: exp(-x)")
29
30 plt.show()
```

3.2.2 Sharing Axis-ticks

Here, same y-axis ticks (i.e. [-3, 2]) are used for two subplots as illustrated in Fig. 3.3 using Listing 3.3. In the listing, line 15 and 16 create two subplots. Further, line 15 contains 'sharey' parameter which sets ticks in the y-axis of subfig2 equal to subfig1.

Listing 3.3: Sharing Y Axis ticks, Fig. 3.3

```
1  #subplotEx2.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  N=100
6  x=np.arange(N)
7  rn = np.random.randn(N)
8  r = np.random.rand(N)
9
10 # create an object 'fig1'  of figure()
11 fig1 = plt.figure()
12
13 # create two instances of fig1
14 subfig1 = fig1.add_subplot(2,1,1)
```

(continues on next page)

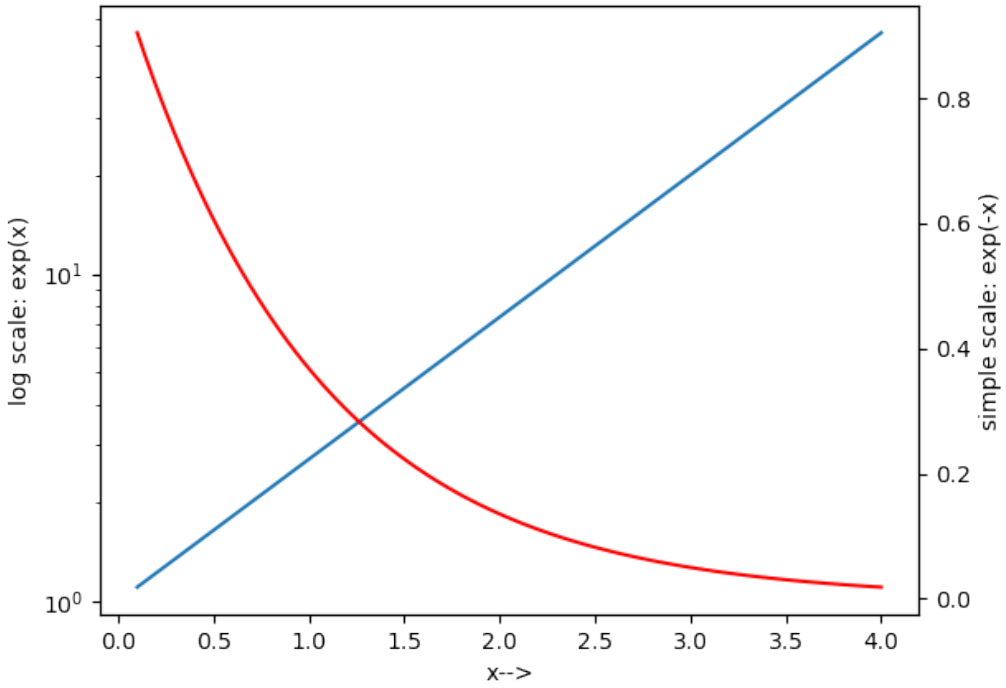

Fig. 3.2: Shared x-axis by two figures, Listing 3.2

(continued from previous page)

```
15 subfig2 = fig1.add_subplot(2,1,2, sharey = subfig1) #share y axis
16
17 # plot figures
18 subfig1.plot(x, rn)
19 subfig2.plot(x, r)
20 plt.show()
```

3.3 Add legend outside the plot

In Listing 3.4, legends are placed outside the figure as shown in Fig. 3.4. It can be quite useful, when we have large number of figures in a single plot. Note that, in line 12, instance of subplot is created directly; whereas in Fig. 3.3, subplot are created using instances of figure(), which require 'add_subplot' command as shown in line 14 and 15 there.

Listing 3.4: Legend outside the plot, Fig. 3.4

```
1  #legendPosEx.py
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  # close all the figures, if open from previous commands
6  plt.close('all')
7
8  ur = np.random.rand(100)
9  nr = np.random.randn(100)
10 x=np.linspace(0,3, 100)
11
```

(continues on next page)

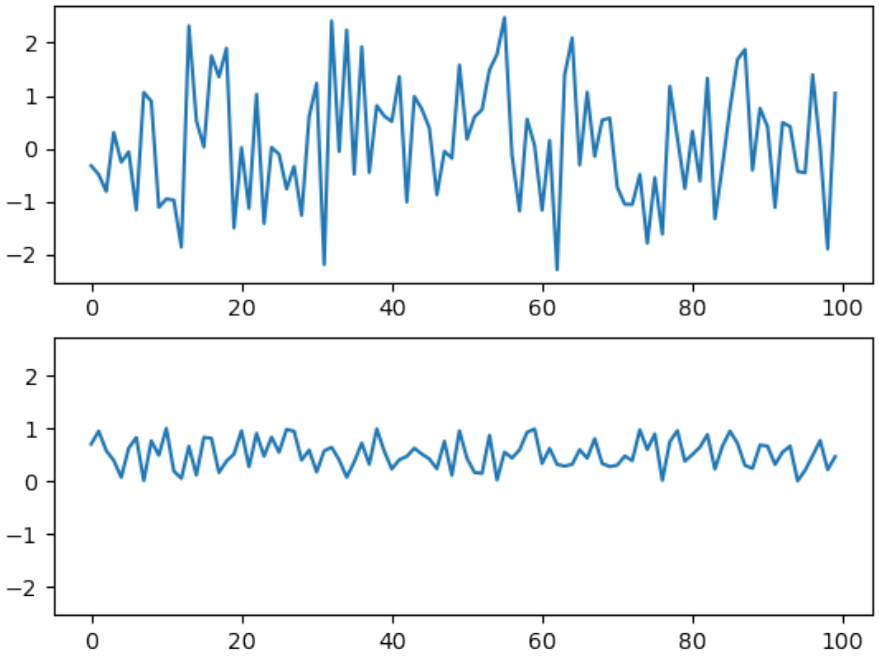

Fig. 3.3: Same Y axis values, Listing 3.3

(continued from previous page)

```
12 ax = plt.subplot(1,1,1)
13
14 ax.plot(x, label="Line")
15 ax.plot(ur, label="Uniform random number")
16 ax.plot(nr, label="Normal random number")
17 ax.set_title("Legend outside the plot")
18
19 # Plot position and shriking to create space for legends
20 box = ax.get_position()
21 ax.set_position([box.x0, box.y0 + box.height * 0.2,
22                  box.width, box.height * 0.8])
23
24 # Add legend below the axis
25 ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),
26           fancybox=True, shadow=True, ncol=2)
27
28 plt.show()
```

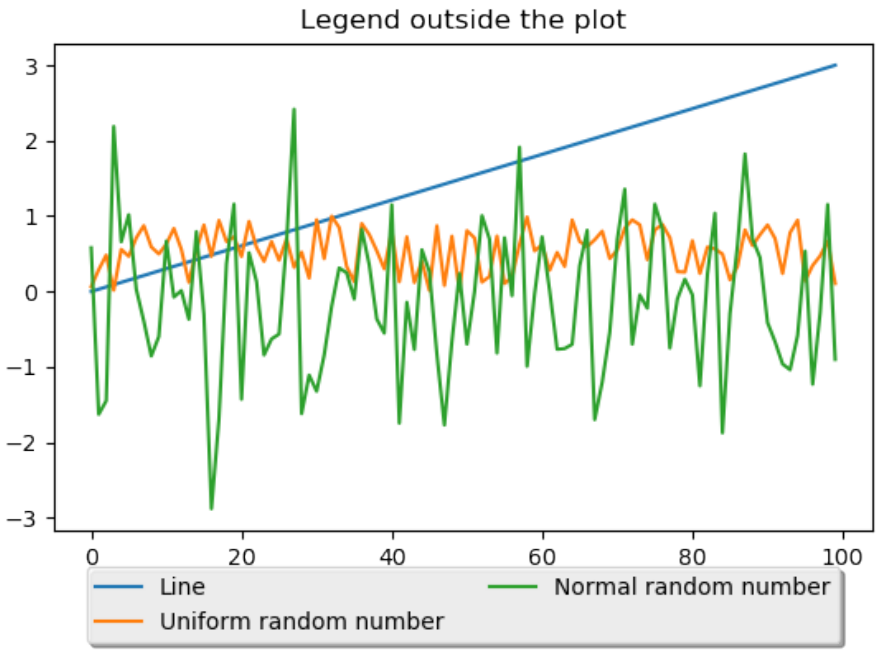

Fig. 3.4: Legend outside the plot, Listing 3.4