

# Matplotlib 簡介

由於人類天生對於圖形化或照片的感知程度要比單純的文字強得多，因此如將數據以圖像方式呈現，便是一門重要的課題，稱為 Data Visualization(DV)。

Python 作為最熱門的一種數據分析應用程式語言，它的數據視覺化函式庫／模組也順理成章成為熱門的使用套件。這其中 Matplotlib 是最具有代表性、使用最廣泛的，雖然有其他更高階的 DV 套件如 seaborn，但也是以 Matplotlib 為基礎來建構的。

要引用 Matplotlib 套件我們可以執行下列命令：

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

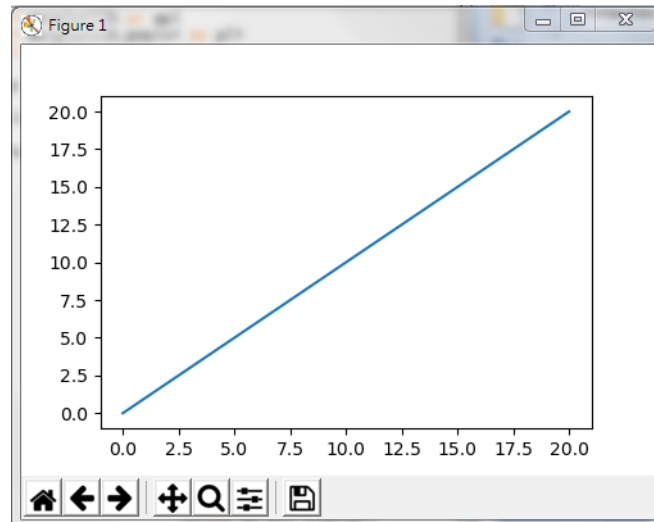
## 1. 先看一個例子

使用 Matplotlib 繪製圖表很容易，在進一步學習之前，我們首先看一個簡單的例子，如何產生一個像樣的圖表。以下是本單元的第一隻程式：

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

x = np.linspace(0, 20, 50)    #產生 50 個數值
plt.plot(x,x)
plt.show()                   #至此真正顯示出畫圖結果
```

執行的結果如下。上述程式中 `plt.plot()` 命令讀入 x 座標與 y 座標的序列（這裡是 NumPy 的陣列資料型態），以產生一個曲線圖，但要等到 `plt.show()` 執行，所有繪圖結果才會呈現出來。此一命令通常放在程式最後面部份執行一次。



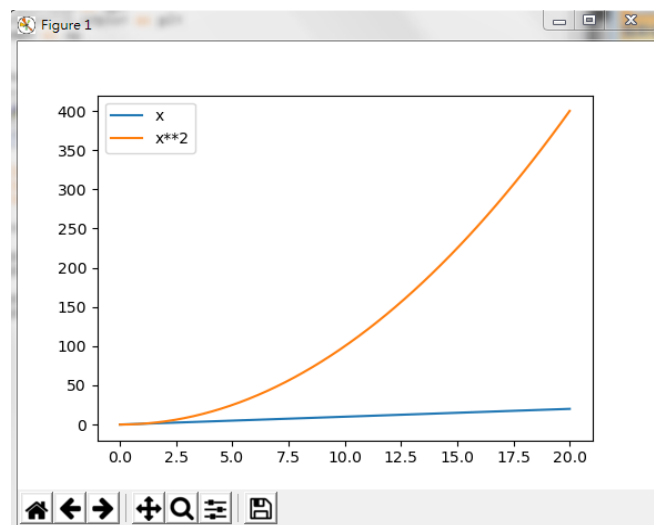
接著我們再加入兩條曲線並且給了 label 標籤值，注意 pyplot 自動上了不同顏色。我們呼叫 `plt.legend()` 指令來產生小看板。

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

x = np.linspace(0, 20, 50)    #產生 50 個數值

plt.plot(x, x, label="x")
plt.plot(x, x*x, label="x**2")
plt.legend()
plt.show()                    # 顯示出來畫圖結果，新版可以不用
```

執行的結果如下：



接著我們再增加一些指令來產生標題和次標到圖表上：

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

x = np.linspace(0, 20, 50)    # 產生 50 個數值

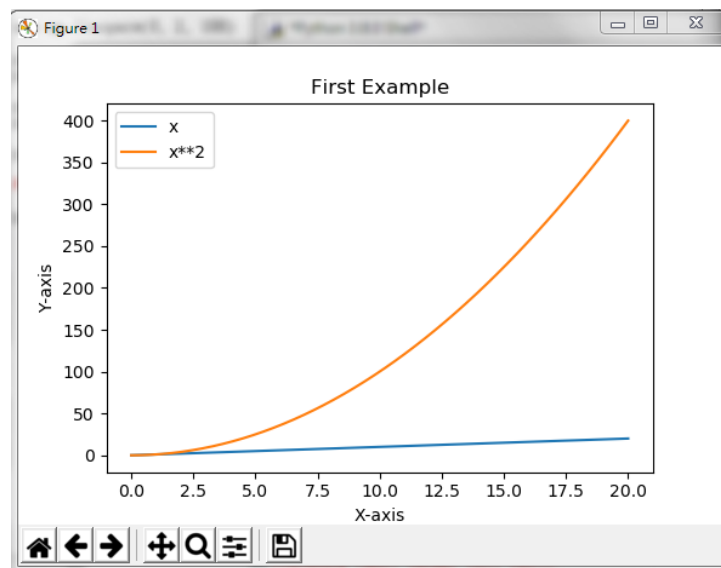
plt.plot(x, x, label="x" )
plt.plot(x, x**2, label="x**2")

plt.title("First Example")
plt.xlabel("X-axis")
plt.ylabel("y-axis")

plt.legend()

plt.show()                    # 顯示出來畫圖結果
```

結果圖表又更豐富了些：



是不是很简单呢？

現在我們把這個圖表轉成圖檔儲存起來，注意：savefig() 要放在 plot() 方法之前，否則後者會創建一個新的空白圖檔，你儲存的圖檔就內容一片空白了。

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

x = np.linspace(0, 20, 50)      # 產生 50 個數值

plt.plot(x, x, label='x' )
plt.plot(x, x**2, label="x**2")

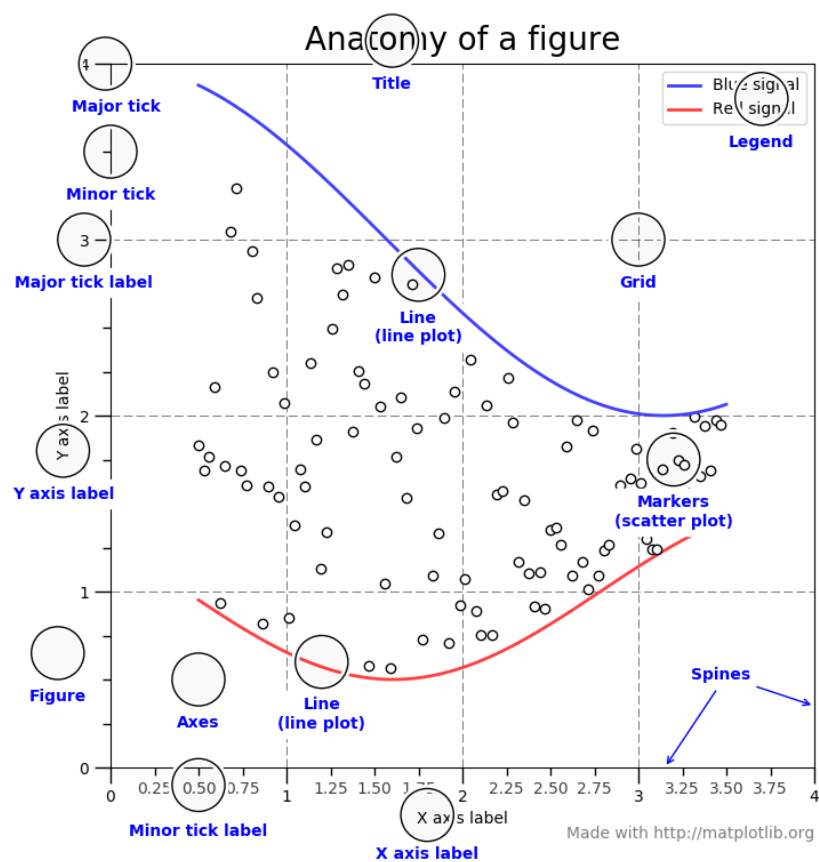
plt.title("First Example")
plt.xlabel("X-axis")
plt.ylabel("y-axis")

plt.legend()
plt.savefig('d:\\image1.png',dpi=100) #顯示之前先儲存圖檔

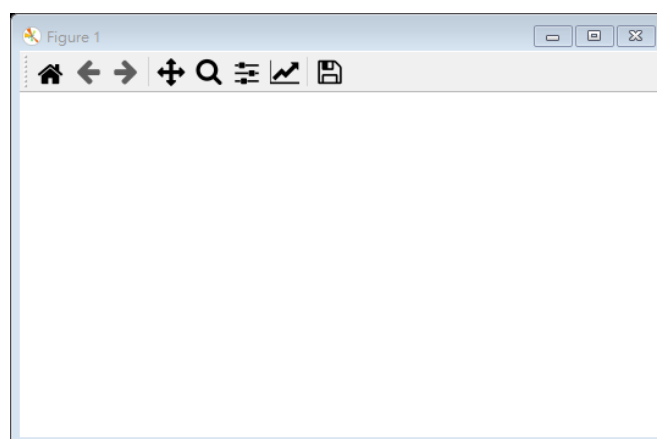
plt.show()

# 另一種做法，如此 savefig()不一定要在 plot()之前
# f = plt.gcf()
# plt.show()
# f.savefig('d:\\image1.png',dpi=100)
```

## 2. 基本概念

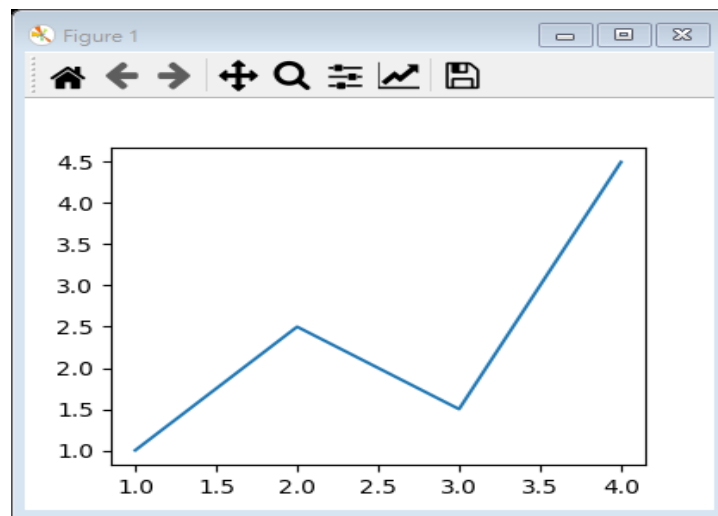


從 matplotlib 官網參考的這張圖很清楚地呈現出一個圖形中可出現名詞和它的定義。基本上，figure 可以視為一個容器（container），就像畫圖的畫板，所有畫圖的元素都包含在裡面，當實際執行程式時，就是指包含圖表的那個視窗，如下圖。

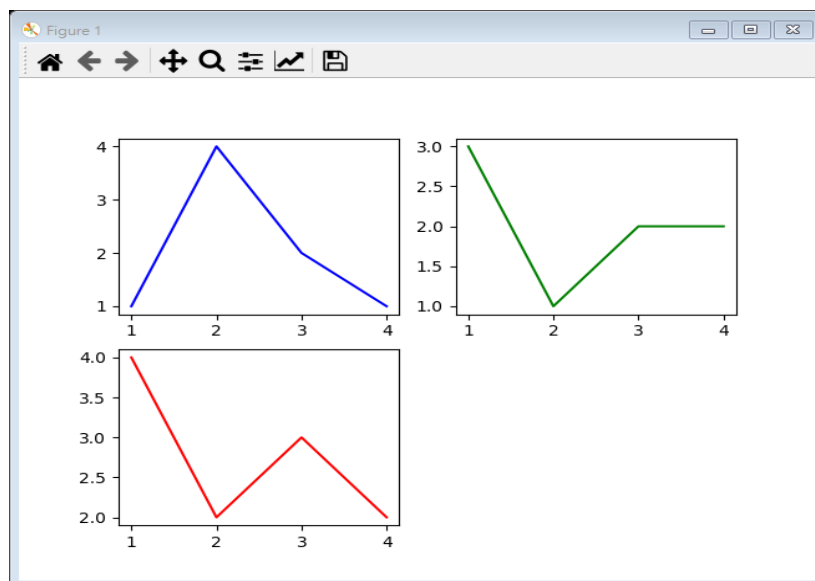


打個比方，它像是用來墊你的畫紙的畫板，而真正你畫的圖紙叫子圖（subplot）

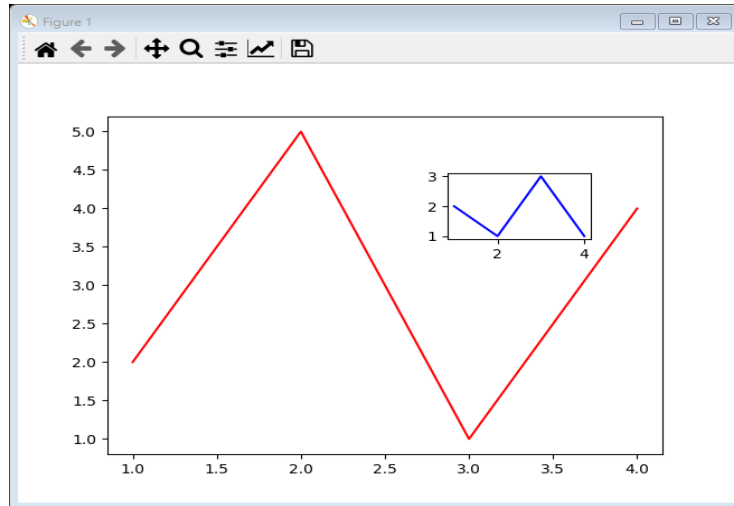
如下圖所示。



一個 figure 可以包含一個以上的子圖，也就是 Axes 或 subplot（就像畫板上的畫紙），這兩個字看起來有點不解差異，可以這麼理解：Axes 或是 Subplot 就像畫紙，才是你畫畫的地方。Subplot 其實是 Axes 的一種，只是不同的 subplot 不能重疊放置，照格子擺放，如下圖；



而 Axes 的圖則無此限。



至於 spines 可以理解為圖表的脊樑線（像圖表上下左右的軸線）；axis 則專指可以加註刻度的兩條軸線：x 軸 與 y 軸。在 axis 上可以設定主要刻度（major ticks）及主刻度之間的小刻度（minor ticks）。圖表可以設定標題（title），軸線與刻度也可以設定說明之文字標記（label）。

另外還有一個重要的東西叫做 artist，artist 字面的意思是「藝術家」，但在這裡其實是泛指圖上所有可看見的元素都是 artist 物件。

### 3. 程式語法風格

基本上，Matplotlib 提供兩種程式撰寫方式，一種是物件導向樣式（OO-style）的寫法，會明白地產生 figure, axes 等等這些物件並使用之=；另一種則是 Pyplot 樣式(Pyplot-style)的寫法，並沒有明白地定義出 figure, axes 這些物件，而是“隱含地”（implicit）方式在引用，程式只有看到使用 pyplot 物件及其函式來產生圖表。這兩種方式都可以採用，但不建議混搭使用。一般而言，後者比較建議在簡單的、交談式操作環境下使用。

我們來看一個 pyplot-樣式的程式例子：

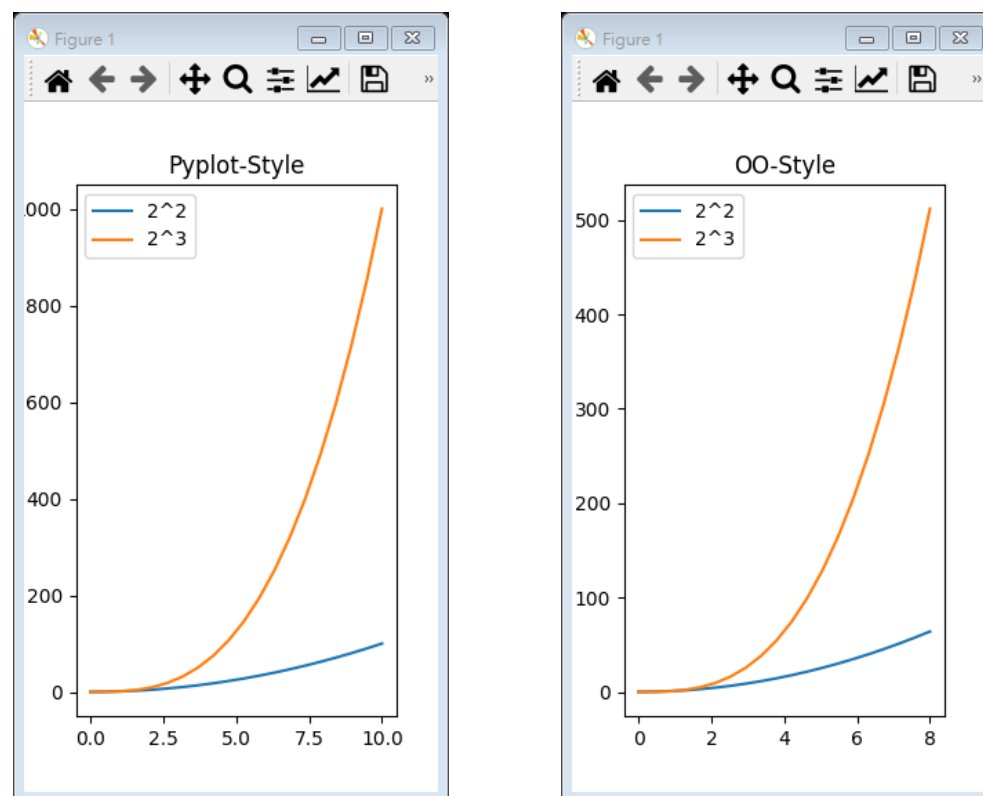
```
import numpy as np
import matplotlib.pyplot as plt
#只有明白使用 pyplot 物件，其餘都是 implicit
x = np.linspace(0, 10, 20)
plt.plot(x, x*x, label='2^2')
plt.plot(x, x*x*x, label='2^3')
plt.title("Pyplot-Style")
plt.legend()
```

OO-樣式的寫法：

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

x = np.linspace(0, 8, 20)
fig= plt.figure(figsize=(3,5))
sp = fig.add_subplot()
sp.plot(x, x*x, label='2^2')
sp.plot(x, x**3, label='2^3')
sp.set_title("OO-Style")
sp.legend()
```

執行結果相似：



從上面的例子我們看到可以採用 `fig= plt.figure()` 這個命令來產生一個 `figure` 的物件並指定給 `fig` 變數，此時便可以用 `fig.add_subplot()` 方法來產生一個子圖物件，進行後續操作。

值得一提的是 `Pyplot` 樣式的寫法並沒有明白地宣告 `figure`, `axes` 這些物件，但如果在程式中需要取得這些物件時要怎麼辦？不用擔心，`Pyplot` 提供了 `gcf()` 跟 `gca()` 函式來取得所需要 `figure` 與 `axes` 物件：



```

import numpy as np
import matplotlib.pyplot as plt
#只有明白使用 pyplot 物件，其餘都是 implicit
x = np.linspace(0, 10, 20)
plt.plot(x, x*x, label='2^2')
plt.plot(x, x*x*x, label='2^3')

plt.title("Pyplot-Style")
f = plt.gcf()    #取得目前 figure 物件
ax= f.gca()      #取得 axes 物件
ax.set_xlabel('X-axis')

plt.legend()

```

#### 4. 單純線圖(Simple Line Plot)

方才的例子就是一個單純的線圖設計，這大概是所有圖表呈現方式裡最簡單的。一個線圖裡面可以畫任意條線，線的顏色 (color)、樣式 (linestyle)、寬度 (linewidth) 也可以改變，以增加可看性。如果未特別指定則 MatPlotLib 會自動配色：

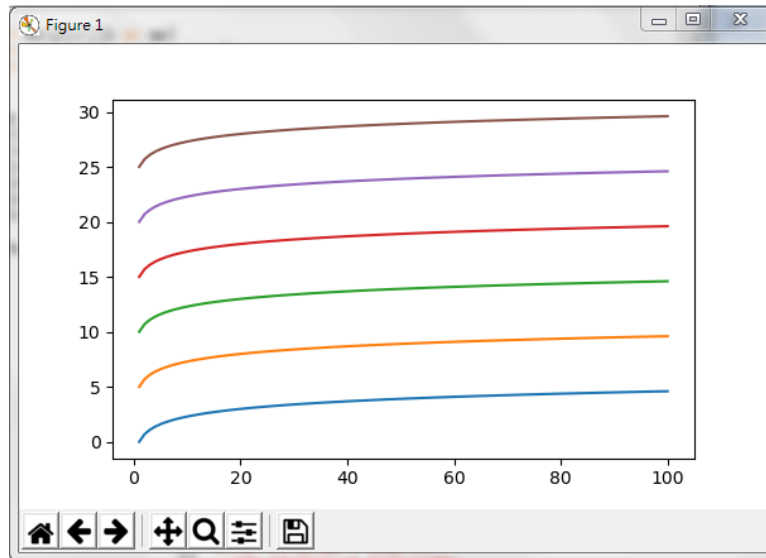
```

import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(1, 100, 100)
plt.figure()    #也可以不寫
plt.plot(x, np.log(x) );
plt.plot(x, np.log(x)+5);
plt.plot(x, np.log(x)+10);
plt.plot(x, np.log(x)+15);
plt.plot(x, np.log(x)+20);
plt.plot(x, np.log(x)+25);

```

結果：



採用 OO-style 寫法如下，其中採用 `plt.subplots()` 函式可以一次產 figure 跟 subplot 物件：

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

x = np.linspace(1, 100, 100)
# fig= plt.figure()
# sp= fig.add_subplot()
fig, sp = plt.subplots()
sp.plot(x, np.log(x) )
sp.plot(x, np.log(x)+5)
sp.plot(x, np.log(x)+10)
sp.plot(x, np.log(x)+15)
sp.plot(x, np.log(x)+20)
sp.plot(x, np.log(x)+25)
```

如果自己配色用 `color` 參數，顏色的表達方式可以有以下數種：

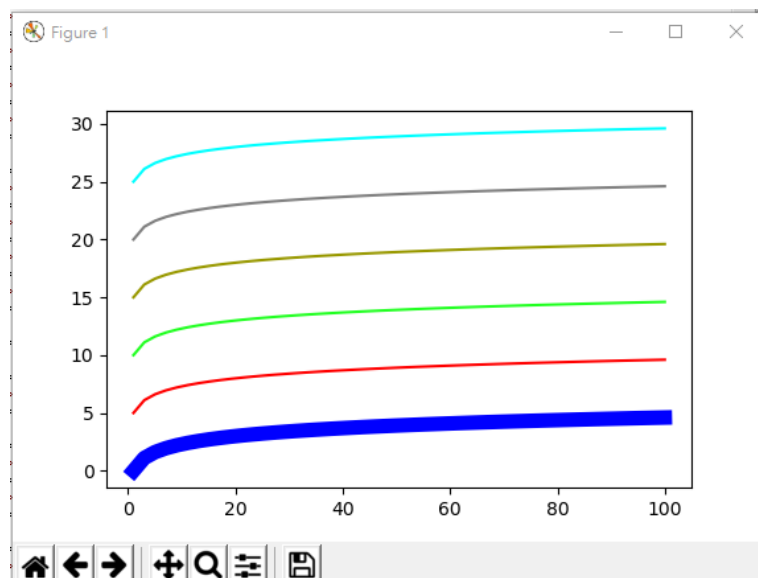
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(1, 100, 100)
plt.plot(x, np.log(x), color="blue", linewidth=8 )
```

```
plt.plot(x, np.log(x)+5 , color="r")          # 縮寫 (rgbcmyk)
plt.plot(x, np.log(x)+10,color="#22FF22")    # RGB Hex code
plt.plot(x, np.log(x)+15,color=(0.6,0.6,0))  # RGB, 值域 0~1
plt.plot(x, np.log(x)+20, color="0.5")      # 灰階亮度值
plt.plot(x, np.log(x)+25,color="aqua")      #HTML 定義顏色名稱

# OO-style
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(1,100,100)
fig, sp= plt.subplots()
sp.plot(x, np.log(x), color="blue", linewidth=8)
sp.plot(x, np.log(x)+5 , color="r")
sp.plot(x, np.log(x)+10, color="#22FF22")
sp.plot(x, np.log(x)+15, color=(0.6,0.6,0))
sp.plot(x, np.log(x)+20, color="0.5")
sp.plot(x, np.log(x)+25, color="aqua")
```

執行結果：
























用 plot() 命令來畫曲線，可以設定的線型有下列幾種：

Line styles	顯示圖形
-------------	------



可設定的標記很多，常用的列舉一些如下：

marker	產生圖案	marker	產生圖案	marker	產生圖案
.		8		x	
,		s		X	
o		P		D	
v		p		d	
^		*			
<		H		-	
>		+		'\$f\$'	

以下是一些例子：

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

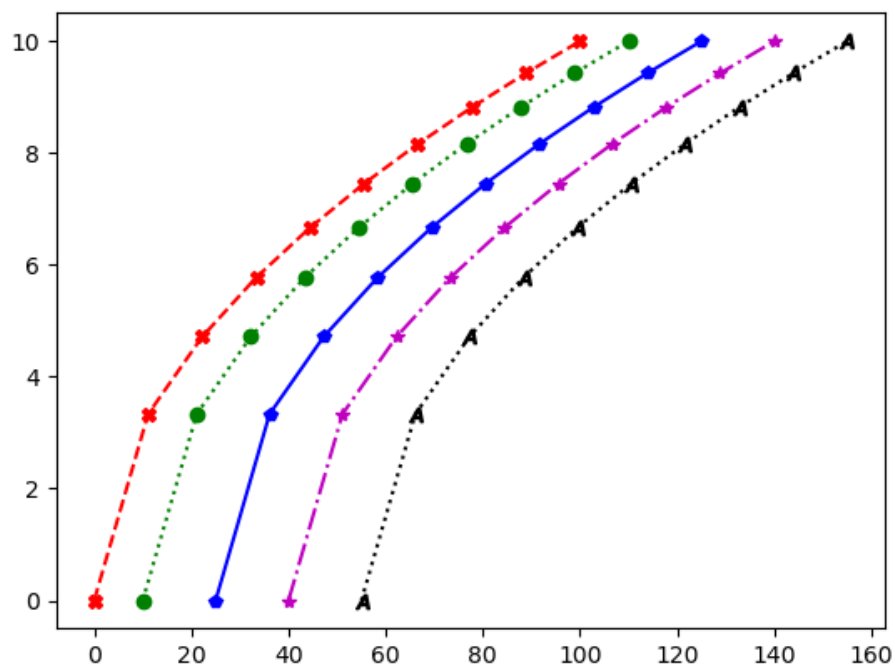
x = np.linspace(0, 100, 10)
y = np.sqrt(x)
fig, sp = plt.subplots()
sp.plot(x, y, color='r', linestyle='--', marker='X')
sp.plot(x+10, y, color='g', linestyle=':', marker='o')
sp.plot(x+25, y, color='b', linestyle='-', marker='p')
sp.plot(x+40, y, color='m', linestyle='-.', marker='*')
sp.plot(x+55, y, color='k', linestyle=':', marker='$A$')
```

樣式設定也可以整合成字串，放在最後一個參數項：

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

x = np.linspace(0, 100, 15)
y = np.sqrt(x)
fig, sp = plt.subplots()
sp.plot(x, y, "gX--")
sp.plot(x+10, y, "ro:")
sp.plot(x+20, y, "b|-")
```

執行結果：



我們可以用 `text()` 方法在圖表上任意位置擺上字串，字串之樣式亦可以設定；也可以用一個方形外框 `bbox` 將字串框起來，且看下例：

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

a = np.linspace(0,21, 100)

b = np.cos(a)

plt.plot(a,b, linewidth=1, color='#FF5555')
plt.title('Show texts')

plt.text(3, -0.85, "min value", fontsize=15, color='b')
plt.text(7, 0.85, "max value",
        bbox= dict(facecolor='#99EEAA', alpha=0.5) )
```

```
#OO-Style
import numpy as np
import matplotlib.pyplot as plt

a = np.linspace(0,21, 100)
```

```

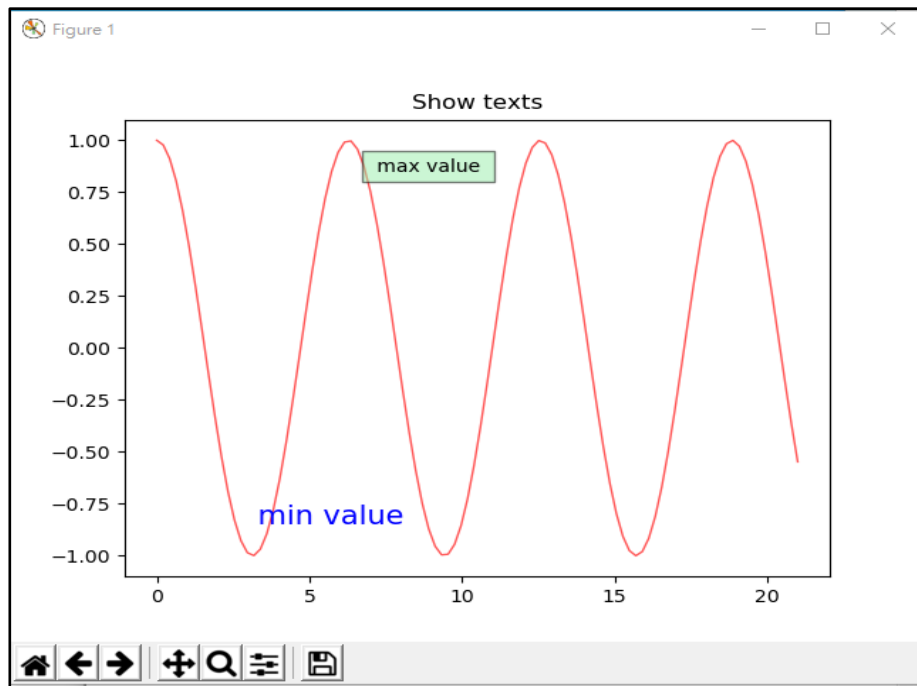
b = np.cos(a)
fig, sp = plt.subplots()
sp.plot(a,b, linewidth=1, color='#FF5555')
sp.set_title('Show texts')

sp.text(3, -0.85, "min value", fontsize=15, color='b')
sp.text(7, 0.85, "max value",
bbox= dict(facecolor='#99EEAA', alpha=0.5) )

```

注意 subplot 物件是用 set\_title()來設定標題。

執行結果：



Plot()提供一種簡便的表示方式，可以用一個指令進行多線段同畫：

```

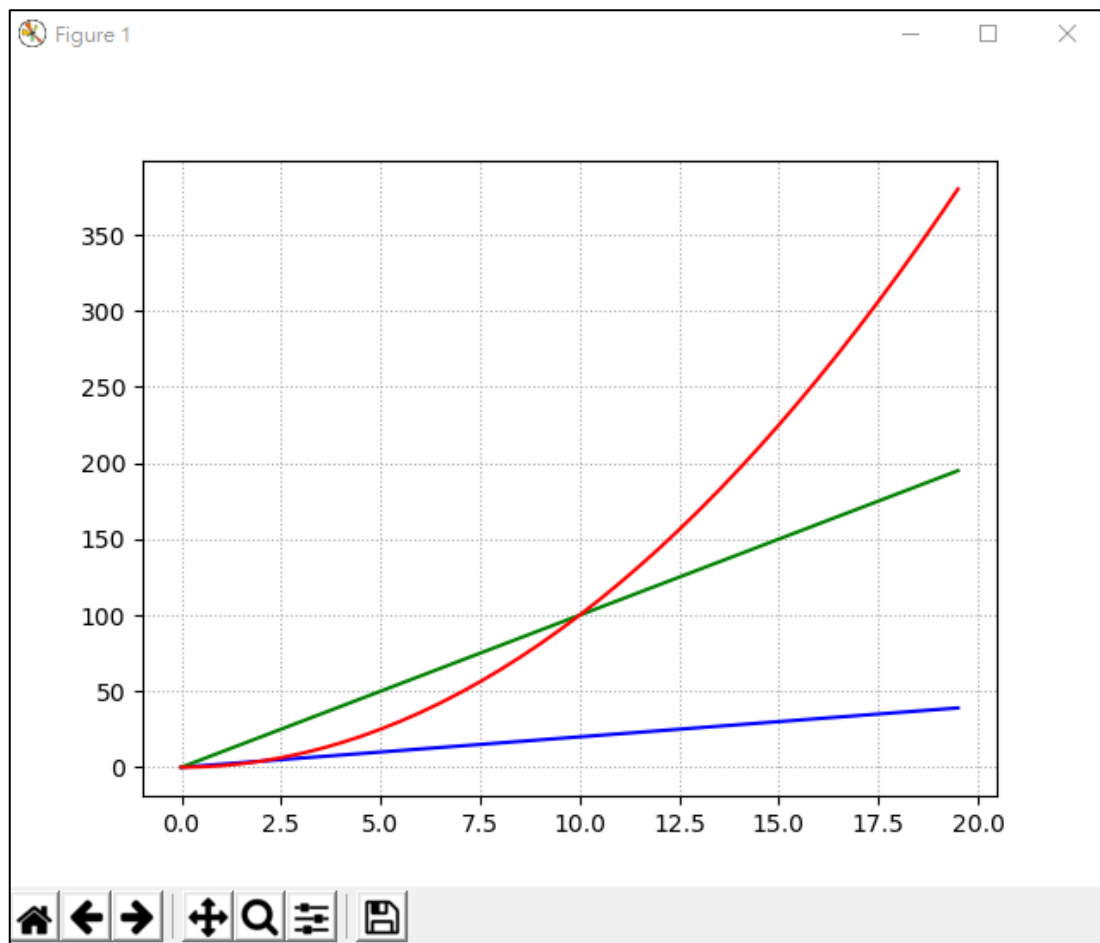
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 20, 0.5)
fig, sp= plt.subplots()
sp.plot(x, x*2, "b-",

```

```
x, x*10, "g-",  
x, x*x, "r-")    # x 座標，y 座標， 直線屬性值，三個一組  
  
sp.grid( color= "#aaaaaa" , linestyle="dotted", linewidth= 0.75)
```

畫出三條線段結果如下：



到目前為止我們畫的線圖都是平滑線段，也就是說相鄰取樣點之間都是走最短距離線段。其實 `plot()` 函式還可以定義其他種的樣式型態，如下例印出了 `step` 函式之三種：

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



```

x = np.arange(0, 24, 1.5)
y = np.sin(x / 3)
plt.figure(figsize=(6,6))
plt.plot(x, y + 4, drawstyle='steps', label='Previous')
plt.plot(x, y + 4, color='k', alpha=0.3)

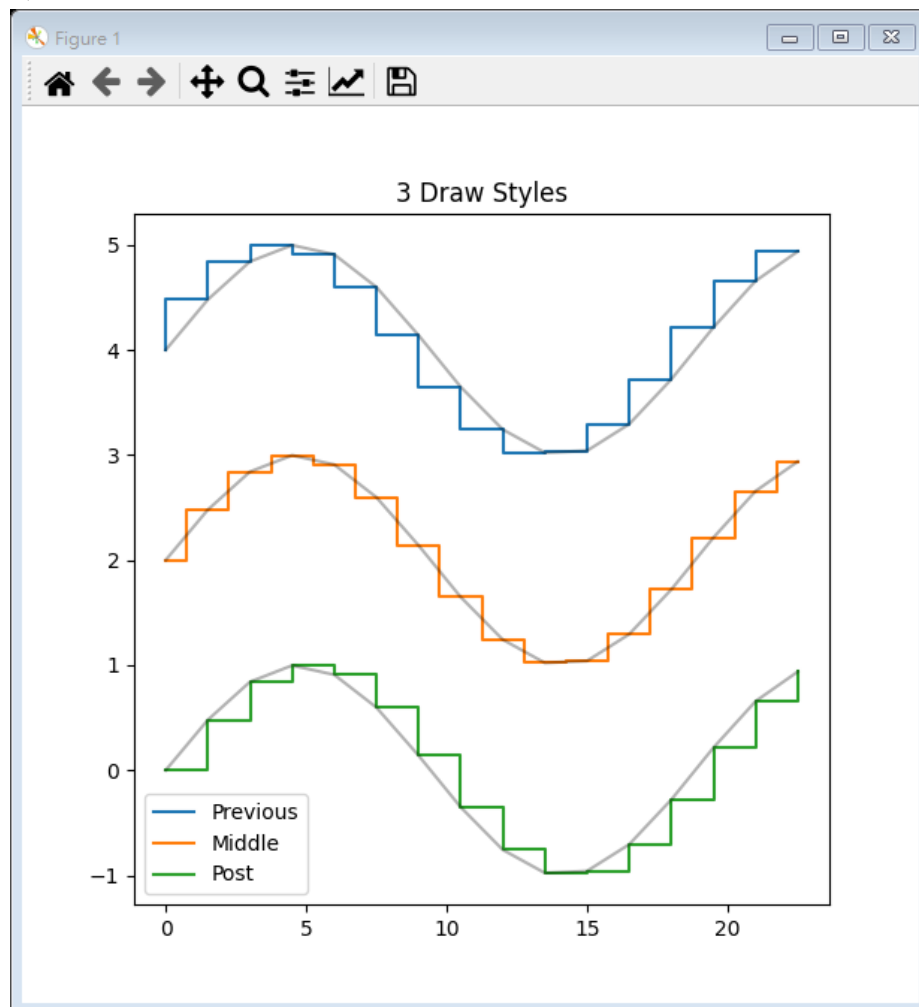
plt.plot(x, y + 2, drawstyle='steps-mid', label='Middle')
plt.plot(x, y + 2, color='k', alpha=0.3)

plt.plot(x, y, drawstyle='steps-post', label='Post')
plt.plot(x, y, color='k', alpha=0.3)

plt.legend( )
plt.title('3 Draw Styles')

```

執行結果：



在 `plot()` 函式中的 `drawstyle` 參數決定了 step 函式是領先 (steps)，置中 (steps-mid)，還落 (steps-post) 後於原來之平滑線。

## 5. 直方圖

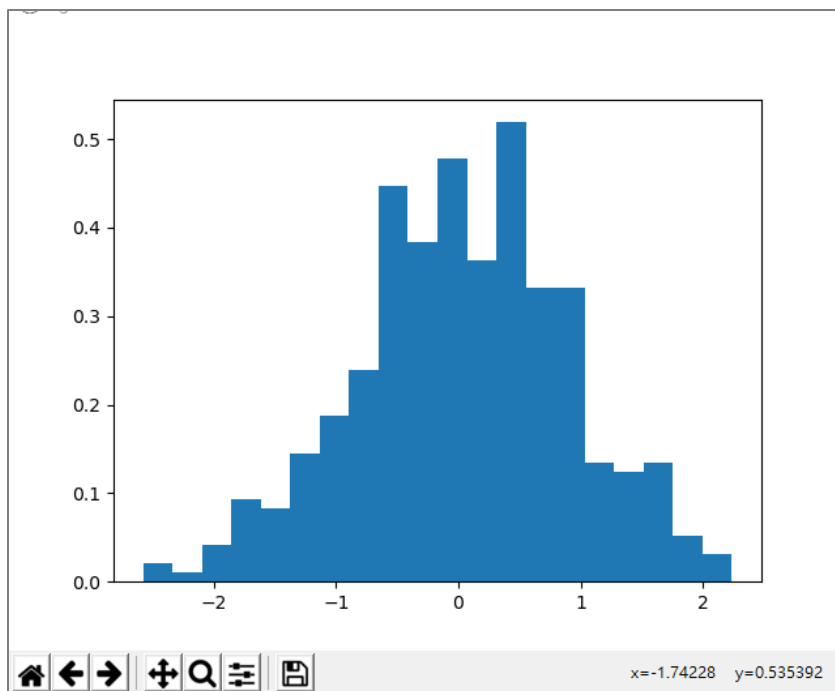
Matplotlib 提供非常多種圖表呈現形態，其中一種是直方圖 (histogram)。我們可以使用 `plt.hist()` 方法來產生直方圖。以下是一個例子，我們用 `np.random.normal()` 函數來產生常態分佈隨機亂數值：

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

x = np.random.normal(size = 400)    #隨機產生 400 個數值
plt.hist(x, bins=20)               #20 個分群

#OO-Style
import numpy as np
import matplotlib.pyplot as plt
x = np.random.normal(size = 400)
fig, sp= plt.subplots()
sp.hist(x, bins=20)
```

執行結果：



透過設定型態(histtype)、透明度(alpha)可以設計出很漂亮的圖表，以下是一個例子：

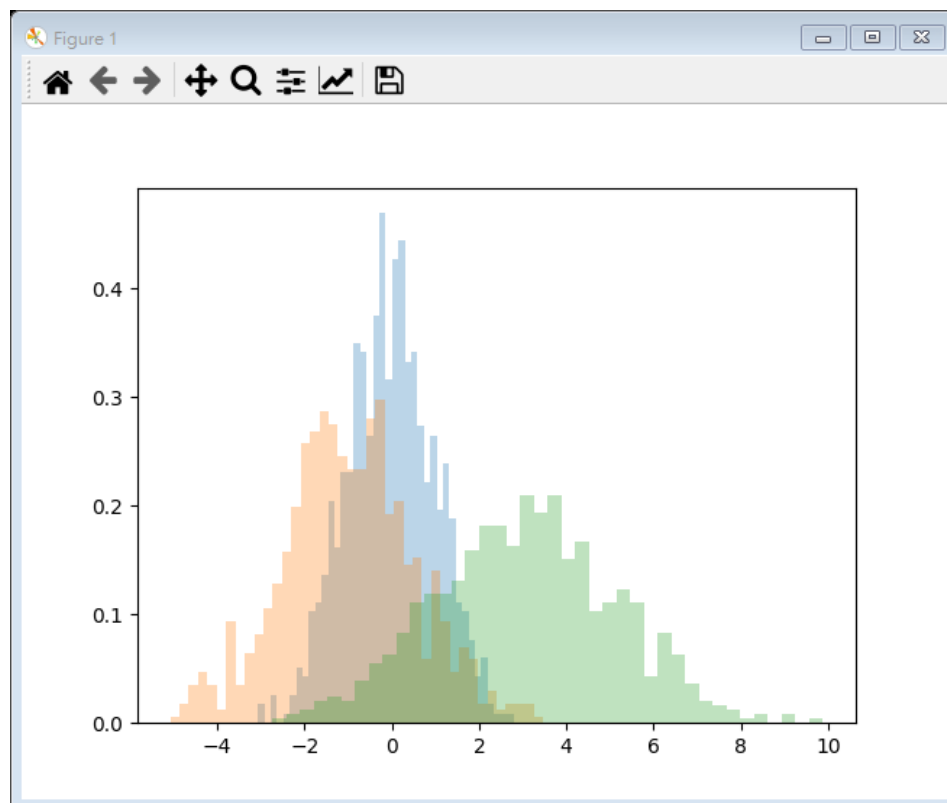
```
import numpy as np
import matplotlib.pyplot as plt

x1 = np.random.normal(0, 1, 800)    #常態分佈的隨機亂數 800 個
x2 = np.random.normal(-1, 1.5, 800)
x3 = np.random.normal(3, 2, 800)
fig, ax= plt.subplots()

ax.hist(x1,bins=40,density=True, alpha=0.3,
        histtype= "stepfilled" )
ax.hist(x2, bins=40, density=True, alpha=0.3,
        histtype= "stepfilled" )
ax.hist(x3, bins=40, density=True, alpha=0.3,
        histtype= "stepfilled" )
```

其中 density=True 之設定，將各欄數值除以總數成為機率密度（介於 0 與 1 之間）。

執行結果：



## 6. Bar 圖

如果是要畫縱向長條圖，請使用 `plt.bar()` 方法；如若要產生橫向的長條圖，可以使用 `plt.barh()` 函數

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

x = np.arange(1,5)
y = 10*x
L1 = ['A1', 'A2', 'A3', 'A4']
L2 = ['B1', 'B2', 'B3', 'B4']

plt.subplot(211)
# 水平長條圖
plt.barh(x, width=y, color='#447722', tick_label= L1 )
# plt.yticks(x,L1])    也可以

plt.subplot(212)
# 垂直長條圖
plt.bar(x, height=y, color='#772244', tick_label= L2)
# plt.xticks(x,L2])    也可以

# OO-Style
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(1,5)
y = 10*x

L1 = ['A1', 'A2', 'A3', 'A4']
L2 = ['B1', 'B2', 'B3', 'B4']
fig, (sp1, sp2) = plt.subplots(2,1)
sp1.barh(x, y, color='#447722', tick_label= L1 )
sp2.bar(x, y, color='#772244', tick_label= L2 )
```

執行結果：



如同單純曲線圖一樣，我們也可以在一個圖表中放好幾組長條圖， 稱之為群組長條圖 (grouped bar chart)。

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

labels= ['G1','G2','G3','G4','G5' ]
A= [ 6, 14, 4, 22, 12]
B= [14, 32, 18, 5, 25]

x = np.arange(5)          # label 位置
width = 0.25              # 直條的寬度
plt.figure()              # 可以不用寫
plt.bar(x-width/2, A, width, label='A')
plt.bar(x+width/2, B, width, label='B')
plt.legend()
```

```
#OO-Style
import numpy as np
import matplotlib.pyplot as plt

labels= ['G1','G2','G3','G4','G5' ]
```

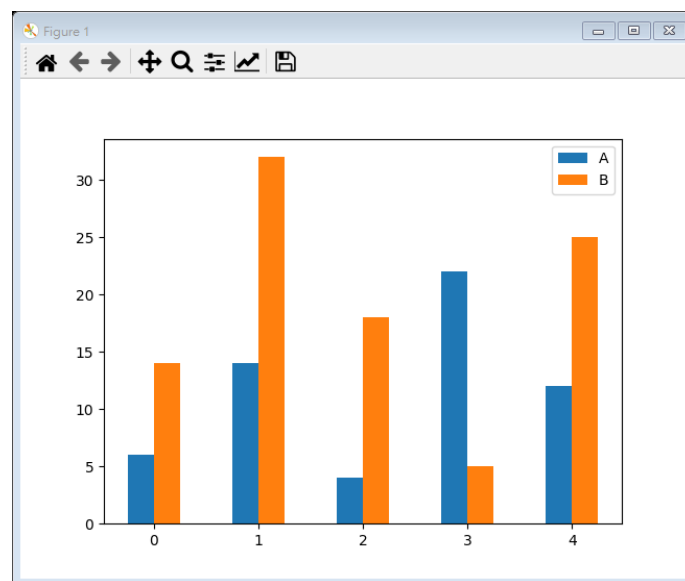
```

A= [ 6, 14, 4, 22, 12]
B= [14, 32, 18, 5, 25]

x = np.arange(5)      # label 位置
width = 0.25          # 直條的寬度
fig, sp=plt.subplots()
sp.bar(x-width/2, A, width, label='A')
sp.bar(x+width/2, B, width, label='B')
sp.legend()

```

結果：



有些時候，我們需要把幾組直方圖重疊以呈現比較結果，其中 bottom 參數用來設定各個長條圖案的起始點：

```

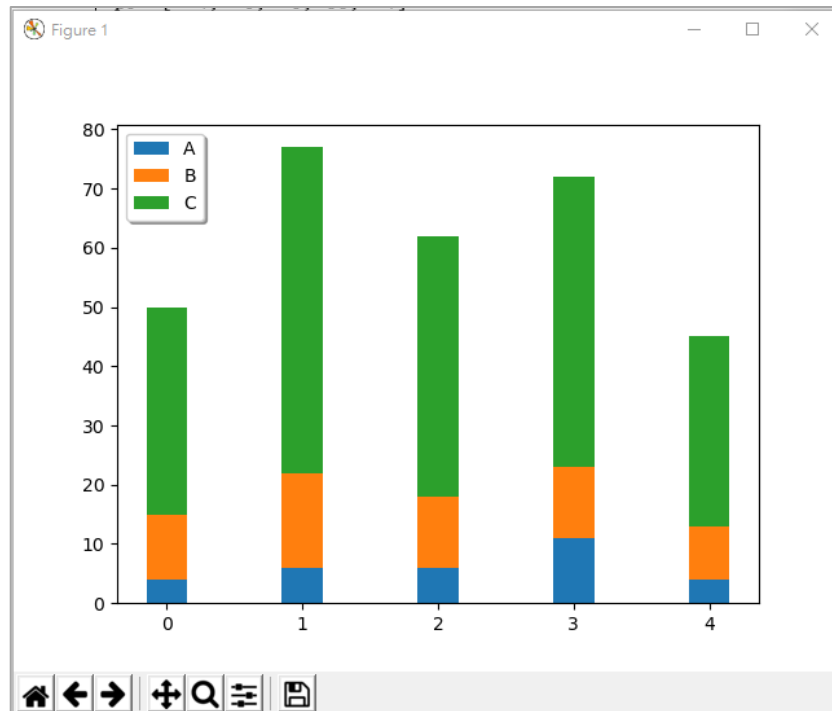
import numpy as np
import matplotlib.pyplot as plt

p1= np.array([ 35, 55, 44, 49, 32])
p2= np.array([ 15, 22, 18, 23, 13])
p3= np.array([4, 6, 6, 11, 4])
ind = np.arange(5)
width =0.3
fig, sp= plt.subplots()
sp.bar(ind, p3, width, label='C')
sp.bar(ind, p2, width, label='B', bottom= p3)
sp.bar(ind, p1, width, label='A', bottom= p2)

```

```
#plt.legend()
sp.legend( ['A','B','C'], loc= 'upper left', shadow=True)
```

顯示出來的結果：



## 7. Pie 圖

要畫出圓餅圖採用 `pie()` 方法，第一個參數是各個部份佔的比例。

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

items = ['A','B','C','D','E','F']
share = [ 15, 20, 40, 10, 7, 5 ]

standout = ( 0, 0, 0, 0.1, 0, 0 )    #只突顯第 4 片

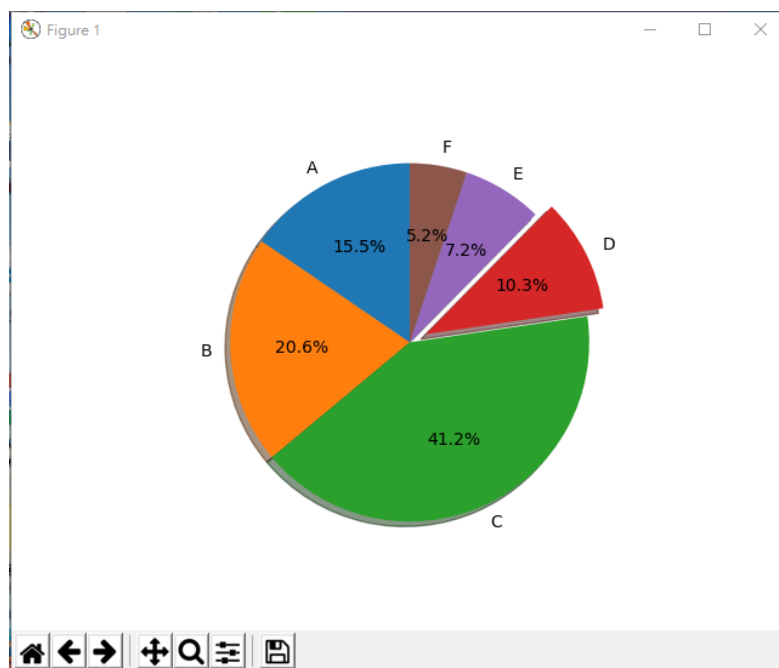
plt.pie(share, labels=items, explode=standout, autopct=
        '%1.1f%%', shadow=True, startangle=90)
```

```
#OO-Style
import numpy as np
import matplotlib.pyplot as plt

items = ['A','B','C','D','E','F']
share = [ 15, 20, 40, 10, 7, 5 ]

standout = ( 0, 0, 0, 0.1, 0, 0 )    #只突顯第4片
fig, sp= plt.subplots()
sp.pie(share, labels=items, explode=standout, autopct=
'%1.1f%%', shadow=True, startangle=90)
```

得到的結果:



## 8. 圓環圖(Donut chart)

我們可以用 pie 圖來設計產生圓環圖(donut 圖)的效果，作法就在 pie 圖中間挖個小圓，其中一種方式是用 `wedgeprops` 參數的設定，舉例如下：

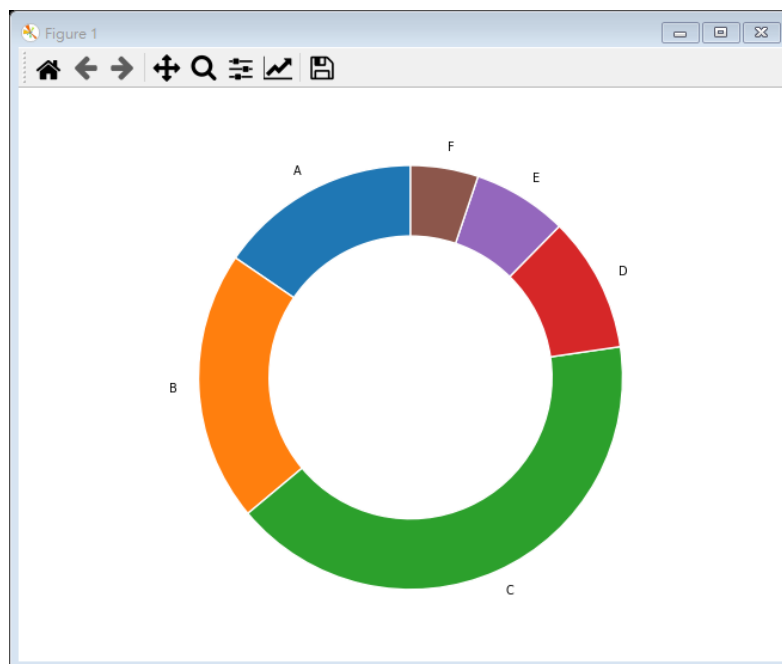


```
#OO-Style
import numpy as np
import matplotlib.pyplot as plt

items = ['A', 'B', 'C', 'D', 'E', 'F']
share = [ 15, 20, 40, 10, 7, 5 ]

fig, sp= plt.subplots()
sp.pie(share, labels=items, radius=1.2,
      wedgeprops=dict(width= 0.4, edgecolor='w') ,
      startangle=90)
```

執行結果：



事實上 wedgeprops 可以設定的參數項目不少，像是設定邊線顏色、線寬等等，參數內容係採用 dictionary 的資料型態加入。

另外，利用畫出不同半徑的 pie 圖，我們也可以設計多層的環狀圖形，在此用另一種變通的方式：最內層用一個“退化的”pie 圖（亦即全圖只有一份，未分切）來產生白色的圓餅：

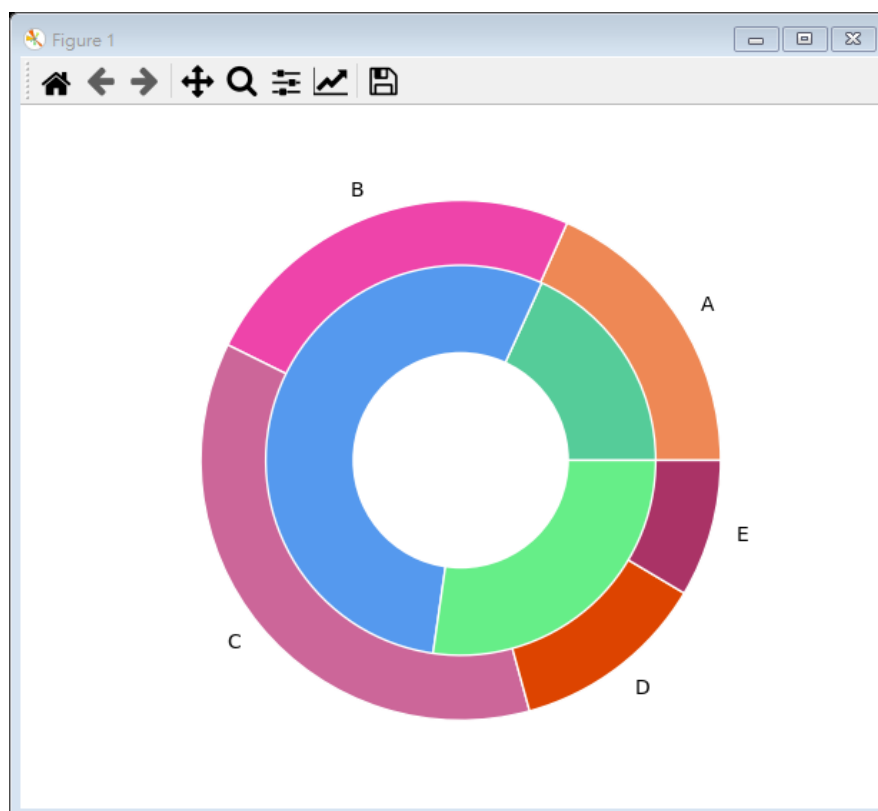
```
import numpy as np
import matplotlib.pyplot as plt
```

```

items = ['A','B','C','D','E' ]
share1 = [ 15, 20, 30, 10, 7]
c1= ['#EE8855','#EE44AA','#CC6699','#DD4400','#AA3366']
c2=[ '#55CC99','#5599EE','#66EE88']
share2 = [ 10,30, 15]
fig, sp= plt.subplots()
sp.pie(share1, labels=items, radius=1.2, colors=c1 ,
      wedgeprops={'edgecolor':'w','linewidth':2})
sp.pie(share2, radius=0.9,startangle=0, colors=c2 ,
      wedgeprops={dict(edgecolor='w',linewidth=2)})
# 畫一塊白餅在中心
sp.pie([10], radius=0.5 , colors=['w'])

```

執行結果如下：



## 9. 堆疊圖(stack chart)

在一張圖上畫出多個折線是常見的圖表呈現方式， 但有時我們需要在線間的區間著上顏色，以強化視覺效果。 要達到這種目的可有不同之作法。 例如採用 `stackplot()` 函式：

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

fig, ax = plt.subplots(figsize=(7,3))
#撤掉右、上之脊線
ax.spines[['right','top']].set_visible(False)

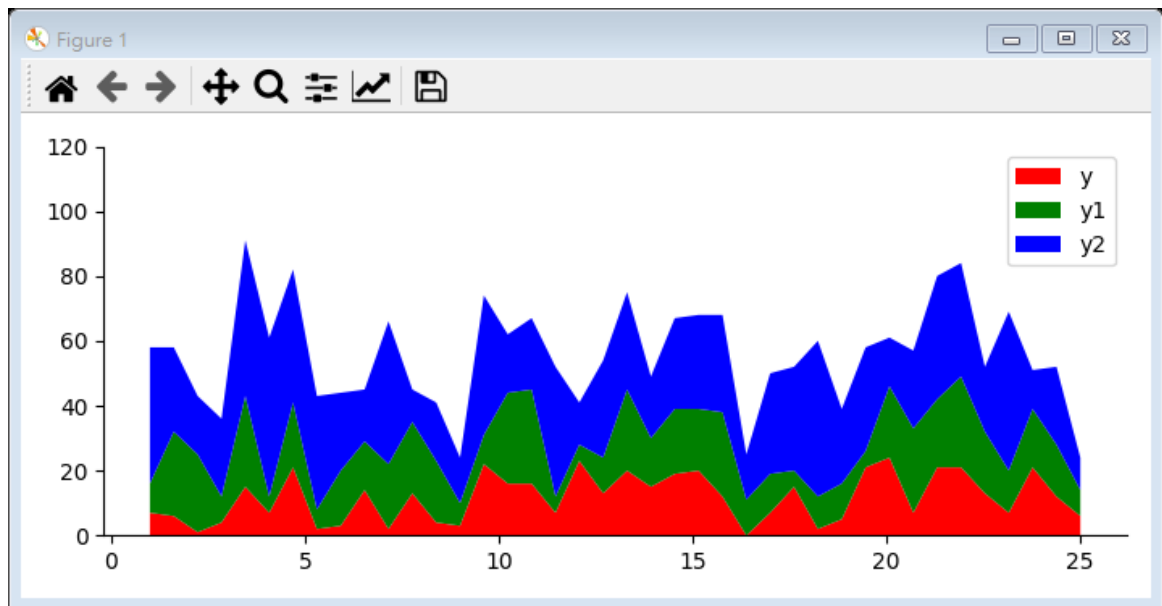
ax.set_ylim(0, 120)

x = np.linspace(1, 25, 40)
y = np.random.randint(0,25,40)
y1 = np.random.randint(5,30,40)
y2 = np.random.randint(10,50,40)

ax.stackplot(x, y, y1, y2, colors=['r','g','b'] ,
             labels=['y','y1','y2'] )
#y 座標值會疊加上去，即 y, y+y1 , y+y1+y2

ax.legend()
```

執行之結果：



另一種方式可以用 `fill_between()` 函式來實現：

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

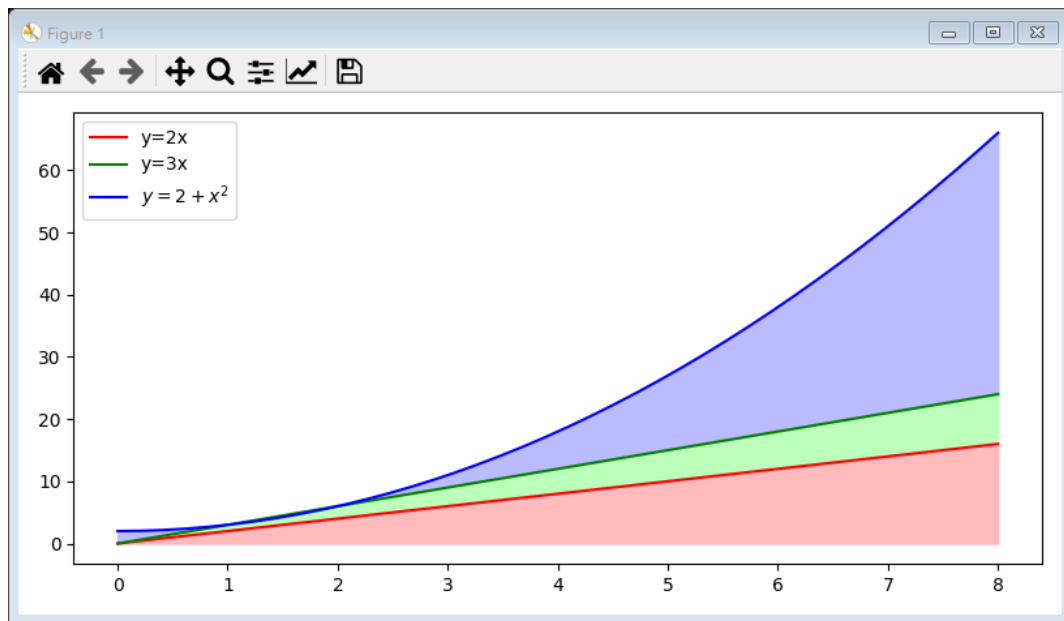
fig, ax = plt.subplots(figsize=(8,4))
x = np.linspace(0, 8, 40)
y = x * 2
y1 = x * 3
y2 = 2 + x * x

ax.plot(x, y, color='r', label='y=2x')
ax.plot(x, y1, color='g', label='y=3x')
ax.plot(x, y2, color='b', label=r'$y=2+x^2$') #正規表示字串

ax.fill_between(x, y2, y1, color='#BBBBFF')
ax.fill_between(x, y1, y, color='#BBFFBB')
ax.fill_between(x, y, 0, color='#FFBBBB')

ax.legend()
```

執行結果如下：



事實上 `fill_between()` 之應用彈性其實是更大的，例如下例，著色區域並無用到軸線：

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

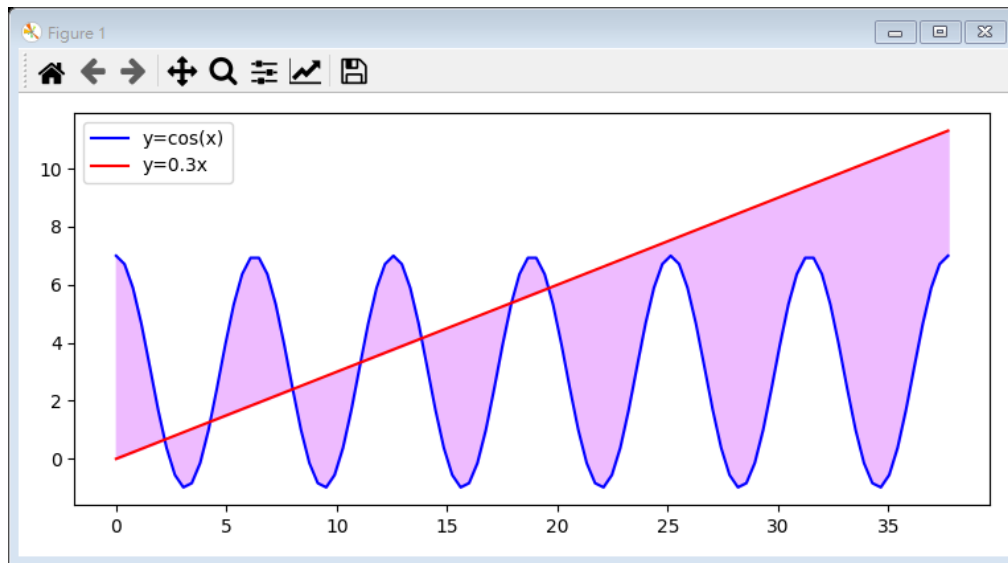
fig, ax = plt.subplots( )

x = np.linspace( 0, 12*np.pi, 100)
y1 = 3+4*np.cos(x)
y2 = x*0.3

ax.plot(x,y1 , color='b' , label= 'y=cos(x)')
ax.plot(x,y2 , color='r' , label= 'y=0.3x')

ax.fill_between(x, y1, y2 , color="#EEBBFF")
ax.legend()
```

執行結果：



## 10. 雜散圖 (Catter plot)

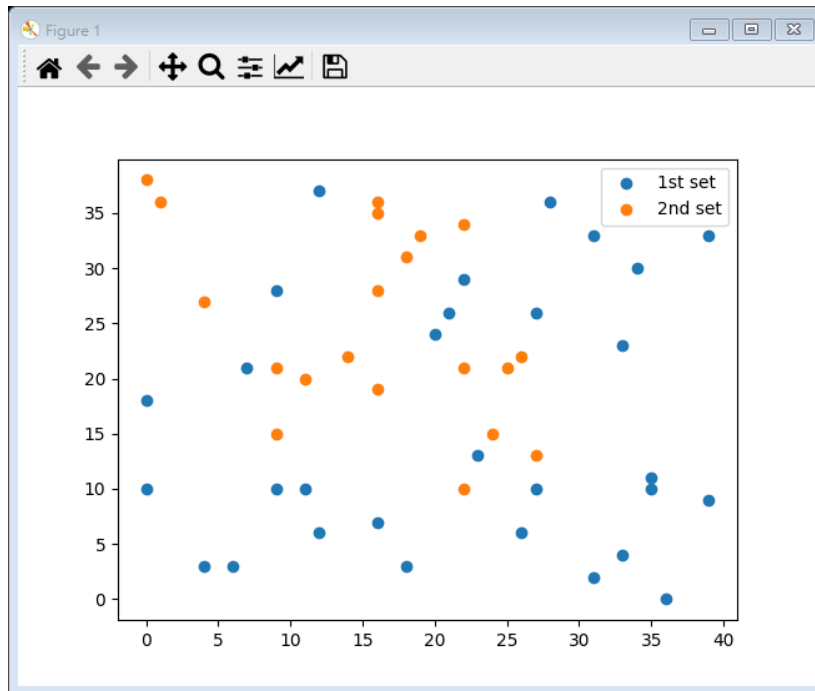
Pyplot 提供了 `scatter()` 函式來讓我們畫出雜散圖表，只要給定 X 座標的陣列和 Y 座標的陣列，就可以產生一系列的點在圖表上。一個簡單的例子如下：

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

fig, ax= plt.subplots()
x = np.random.randint(0,40, 30)
y = np.random.randint(0,40, 30)
ax.scatter(x,y, label="1st set")

x = np.random.randint(0,30, 20)
y = np.random.randint(10,40, 20)
ax.scatter(x,y , label="2nd set")
ax.legend()
```

執行結果如下：



我們也可以指定一組顏色陣列，讓圖表依序選擇點之顏色；也可以設定一個 size 的陣列來給定每個點的大小。前述二陣列的示素個數必須與點的數目相同。如果點數很多我們可以引用某一個 colormap（色圖，即從所有顏色值中預選好的一組顏色子集）然後將引用之索引值存在陣列，指定給 scatter 圖，舉例如下：

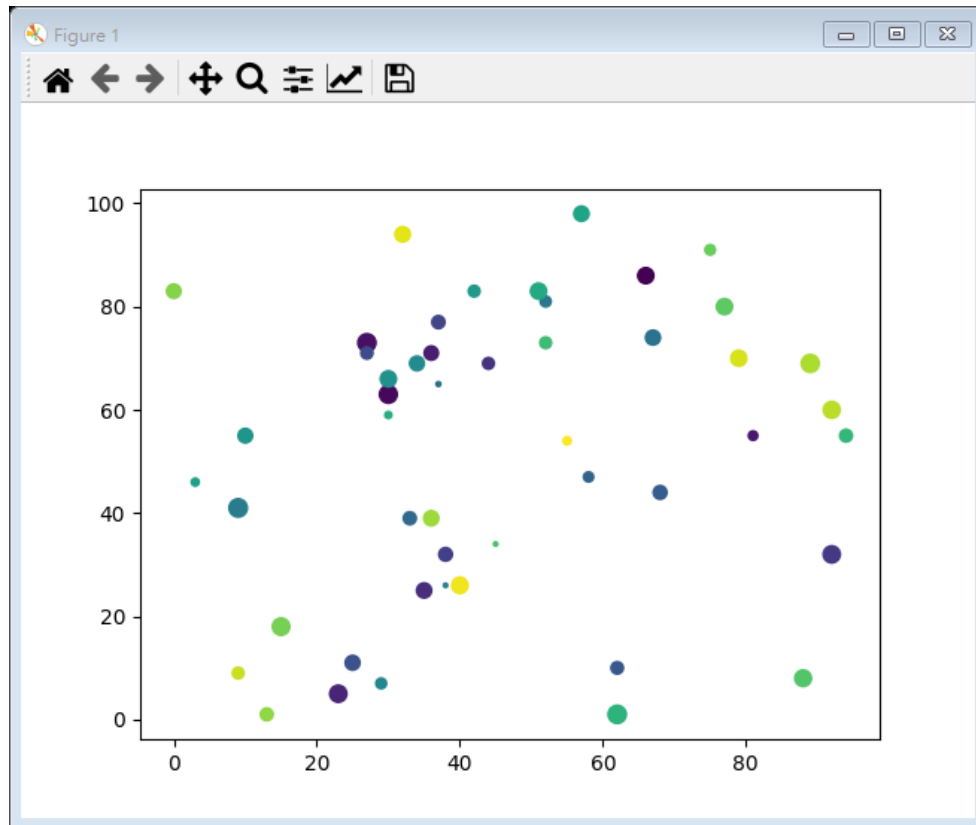
```
import matplotlib.pyplot as plt
import numpy as np

fig,ax= plt.subplots()
clist= np.arange(1,50)
s = np.random.randint(1,80,49)

x = np.random.randint(0,100, 49)
y = np.random.randint(0,100, 49)

ax.scatter(x,y, c=clist, sizes=s, cmap='viridis' )
```

執行結果如，可以看點的顏色與大小都有變化：



## 11. Color Map

在 Matplotlib 中，任何一個色彩可以用 RGB 三色的亮度值來定義，值的大小介於 0~255 之間，例如鮮紅色的值用 16 進位表示是 '#FF0000'。所以整個色彩空間（RGB 分別看成 XYZ 軸的話，每個顏色相當於空間的一個點）將有  $256 \times 256 \times 256$  這麼多種顏色！我們可以根據特定需求，從色彩空間中挑選出一些符合需求的子集合來使用，這就是 colormap。Matplotlib 已經定義了很多 colormap，可選用，它把這些 colormaps 依功能性分成幾大類，每一類都定義了很多色圖：

- sequential colormaps：序列化色圖，著重在特定色調下，逐漸調變亮度與飽和度的結果。又可分一般序列化或是感覺均勻(perceptually uniform) 序列化。這種色圖適合用來表達有順序性關係的資料。
- diverging colormaps：兩端發散的色圖。通常兩端是不同顏色，然後分別向中間調變亮度與飽和度，最後在中間融合成一個顏色。
- cyclic colormaps：循環色圖，色圖兩端之顏色相同，中間之顏色以對稱方式演變至兩端。



- qualitative colormaps：離散化色圖。不是連續調變之色彩。

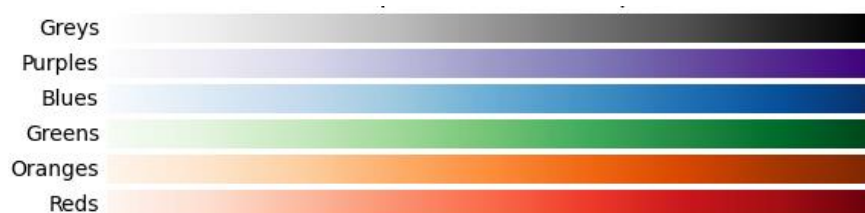
以下是一些色圖的例子（非全部）：

#### (1) sequential colormaps

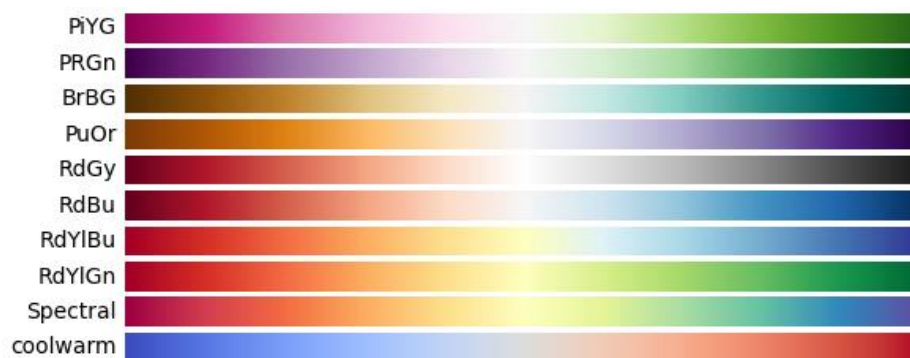
感知均勻序列化：



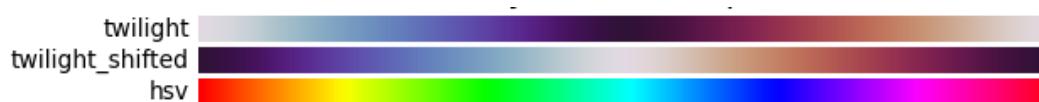
序列化：



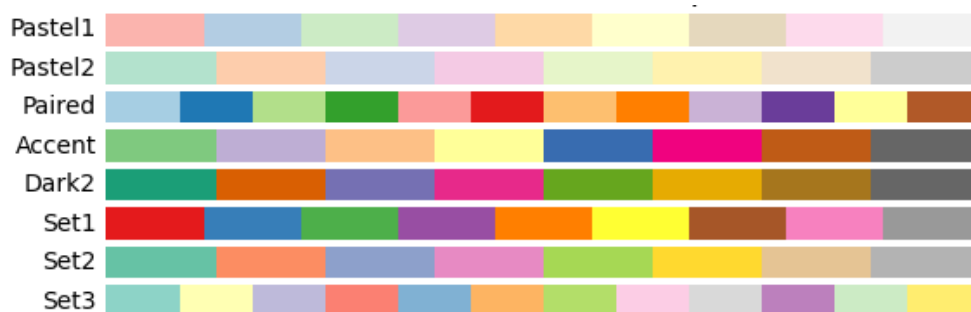
#### (2) diverging colormaps



#### (3) cyclic colormaps



#### (4) qualitative colormaps



(source: <https://matplotlib.org/stable/tutorials/colors/colormaps.html> )

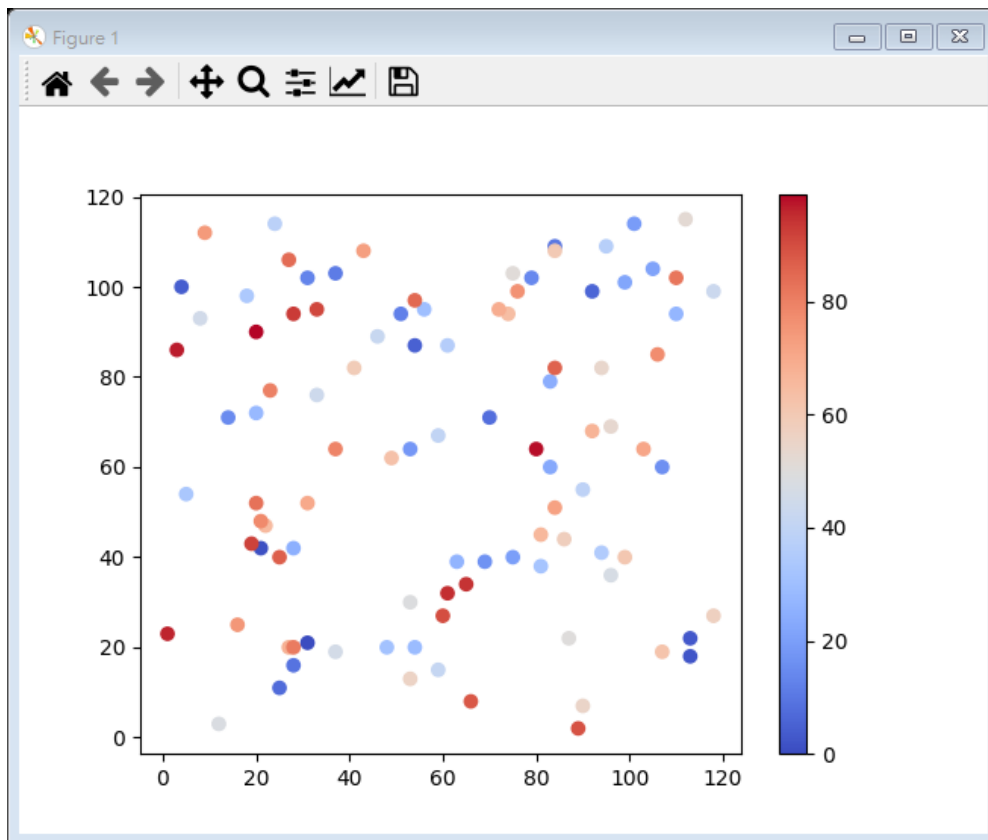
當你的圖表需要用到大量之顏色時，採用 `colormap` 是一個不錯的方法。而且許多 `colormap` 之設計是有考慮人類感受因素，可以適合不同之應用情境。值得一提的，有時我們會綁定一個 `colorbar` 到圖表，用以呈現可以使用的顏色及其代表意義。以下是一個例子：

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

fig, sp = plt.subplots()
clist= np.arange(1,50)
x = np.random.randint(0,120, 100)
y = np.random.randint(0,120, 100)

m= sp.scatter(x,y, c=clist, size=12, cmap='coolwarm')
fig.colorbar(m, ax=sp) #建立並連結色條至 scatter 圖
```

執行之結果如下：



## 12. 全域參數調整

MatPlotLib 的每一個應用專案，所有相關屬性值都會記錄在自有的 matplotlibrc 配置檔案中，我們稱之為 rc settings 或 rc params，它決定整個 Pyplot 圖中幾乎每個屬性的預設值：大小，線寬，顏色和樣式，dpi，軸，軸和網格屬性，文字和字型..等等。所有 rc 參數值都存在名為 rcParams 的字典中。這些參數的影響是全域性的，提供各元素之各種預設狀態值。

在產生圖表時，個別元素的樣式通常是會採各自設定；但如果是想要改變大家的基本樣式，那就可以考慮修改 pyplot.rcParams 字典內的預設值。Rc 參數鉅細靡遺，數量相當多，可以在程式中用 **rcParams.key()** 命令列出全部。我們舉一個參數應用的例子例下：

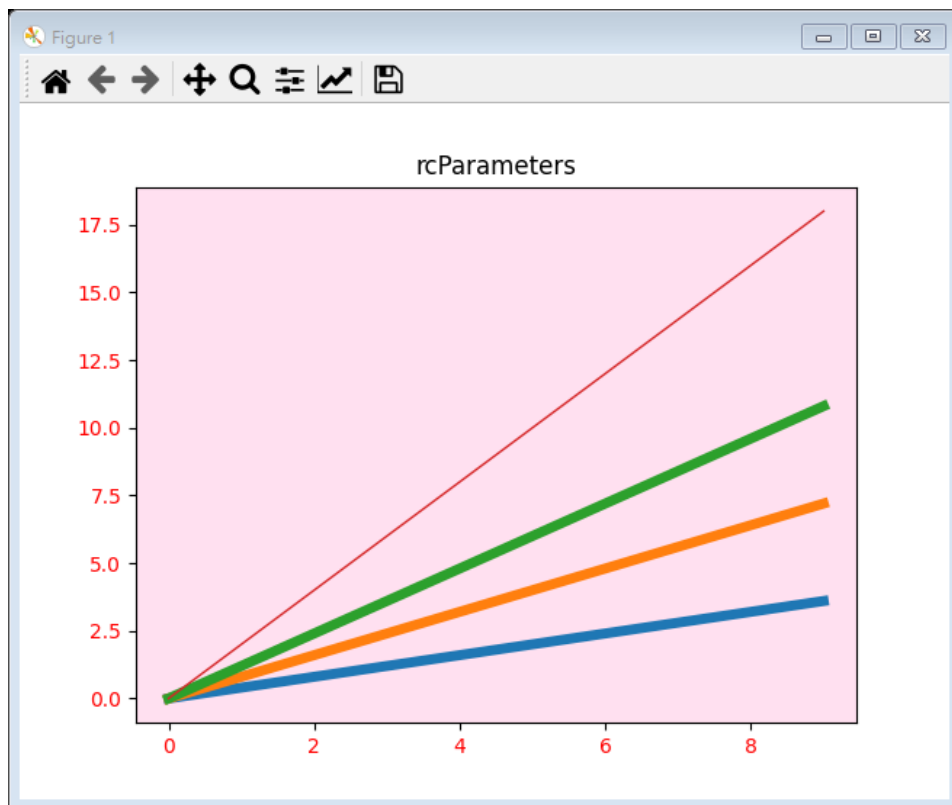
```
import numpy as np
import matplotlib.pyplot as plt

x= np.arange(10)

fig,ax=plt.subplots()
plt.rcParams['lines.linewidth'] = 5      #改變預設線寬
plt.rcParams['axes.facecolor'] = '#FFE5F2'
plt.rcParams['xtick.labelcolor'] = '#FF0000'
plt.rcParams['ytick.labelcolor'] = '#FF0000'

ax.plot(x, 0.4*x)
ax.plot(x, 0.8*x)
ax.plot(x, 1.2*x)
ax.plot(x, 2*x, linewidth=1)      #自訂線寬
ax.set_title("rcParameters")
```

執行結果如下：



### 13. 多圖並呈

Plt-Style 模式下使用 `subplot()` 方法，可以在一個視窗中，同時呈現多個圖表，並指定各個圖的排列位置。舉例如下：

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

x = np.linspace(0, 10, 80)

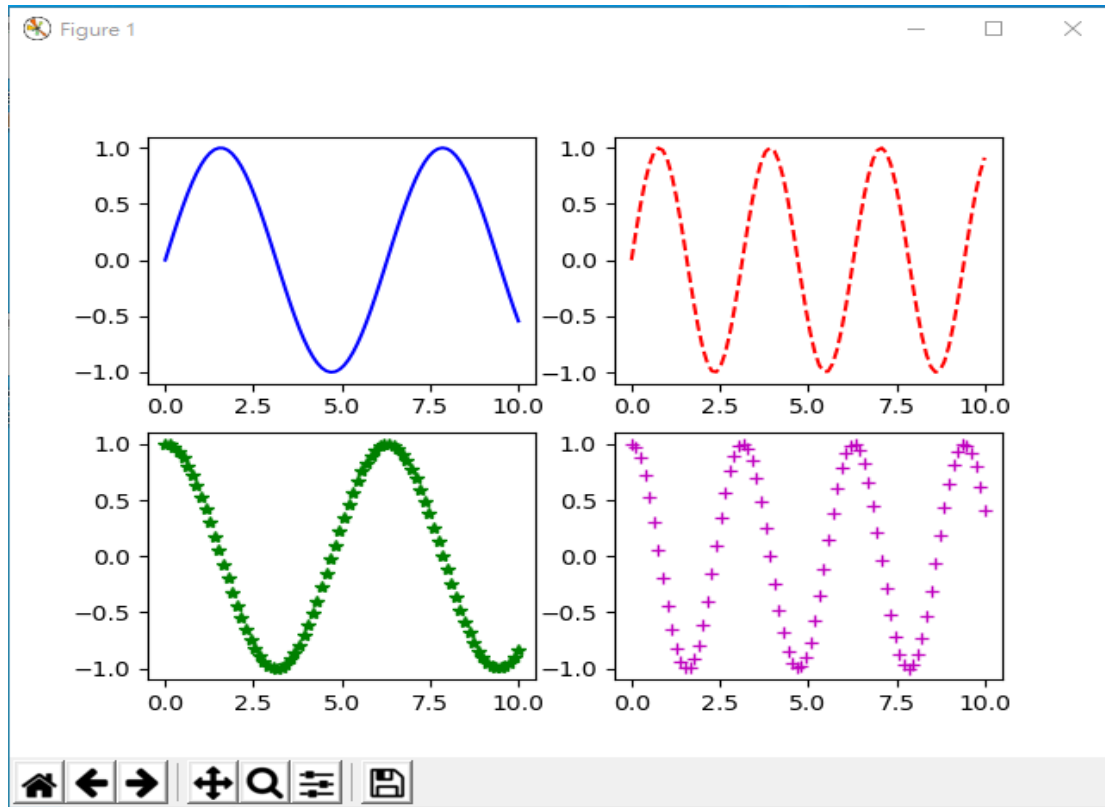
plt.subplot(221)          # 兩列，兩行，第 1 張子圖
plt.plot(x, np.sin(x), 'b-')

plt.subplot(222)          # 兩列，兩行，第 2 張子圖
plt.plot(x, np.sin(2*x), 'r--')

plt.subplot(223)          # 兩列，兩行，第 3 張子圖
plt.plot(x, np.cos(x), 'g*')
```

```
plt.subplot(2,2,4)          # 逗號分開參數也行，第4張子圖
plt.plot(x, np.cos(2*x), 'm+')
plt.show()
```

執行結果：



我們也可以設定多個 figure，每一個 figure 會對應到一個視窗。使用 `plt.figure(n)` 命令切換目前焦點到第 `n` 個 figure。

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

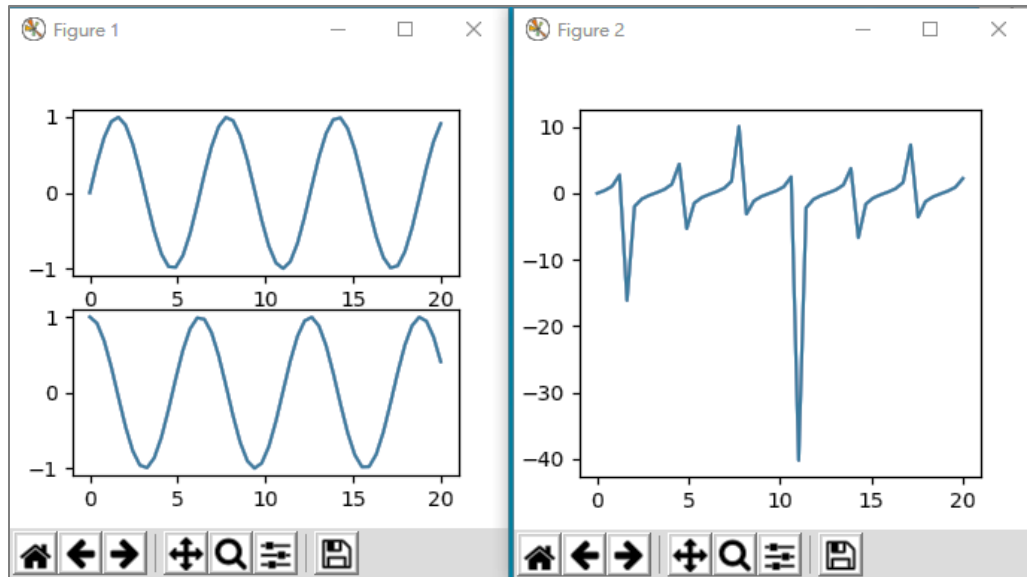
x = np.linspace(0, 20, 50)

plt.figure(1)                # 目前是第1個 figure
plt.subplot(211)
plt.plot(x, np.sin(x))
plt.subplot(212)
plt.plot(x, np.cos(x))
```

```
plt.figure(2)
plt.plot(x, np.tan(x))
```

#目前是第2個 figure

執行結果跳出兩個視窗



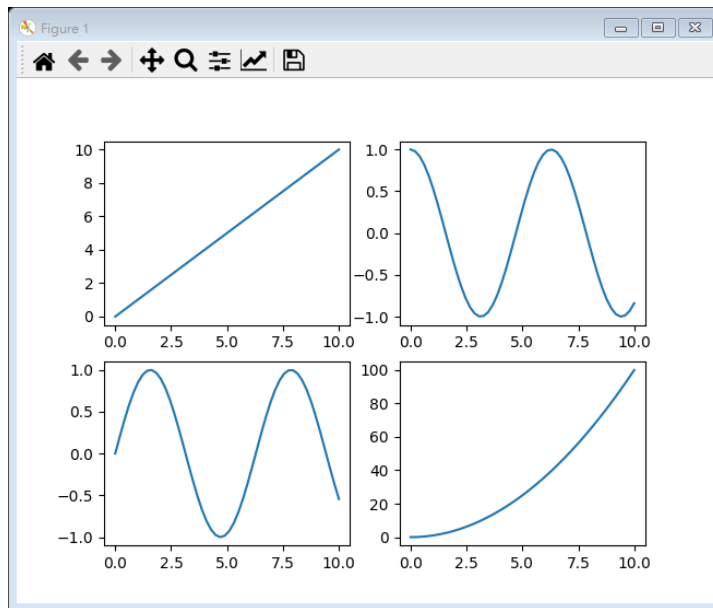
在 OO 模式下， 可以用 `plt.subplots()` 下列方式產生若干子圖：

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

fig1, axes = plt.subplots(2,2)

x= np.linspace(0,10,50)

axes[0,0].plot(x,x)
axes[0,1].plot(x, np.cos(x))
axes[1,0].plot(x, np.sin(x))
axes[1,1].plot(x, x*x)
```



也可以先產生 figure 物件再一個一個加上子圖：

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

x=np.arange(0,10,0.2)
x1= np.arange(6)
y1= np.random.randint(1,50,6)
fig = plt.figure(1)

ax1= fig.add_subplot(221)
ax2= fig.add_subplot(222)
ax3= fig.add_subplot(223)
ax4= fig.add_subplot(224)

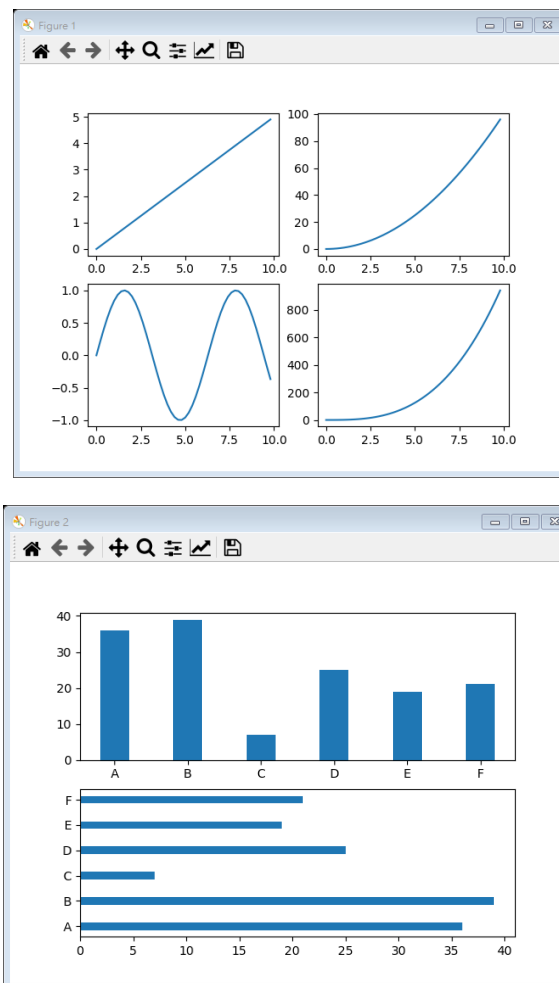
ax1.plot(x, x*0.5)
ax2.plot(x, x*x)
ax3.plot(x, np.sin(x))
ax4.plot(x, x**3)

fig2 = plt.figure(2)

ax5= fig2.add_subplot(211)
```

```
ax6= fig2.add_subplot(212)
ax5.bar(x1, y1, width=0.4)
ax6.barh(x1,y1. height=0.3)
```

執行結果：



## 14. 圖表排版

Pyplot 的子圖是以格柵 (grid) 的概念擺置在 figure 上，先前的列子我們看到的都是每個子圖佔一格，但如果我們希望不同的子圖佔據的格數有所不同時，要如何設計？這其實有不只一種方式來處理，一種方式是用 `pyplot.subplot2grid()` 函式來做，此種方式等於是採用隱含方式同時產生



grid 架構和子圖的擺放位置，格子的行列索引值從 0 開始計算。舉例如下：

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

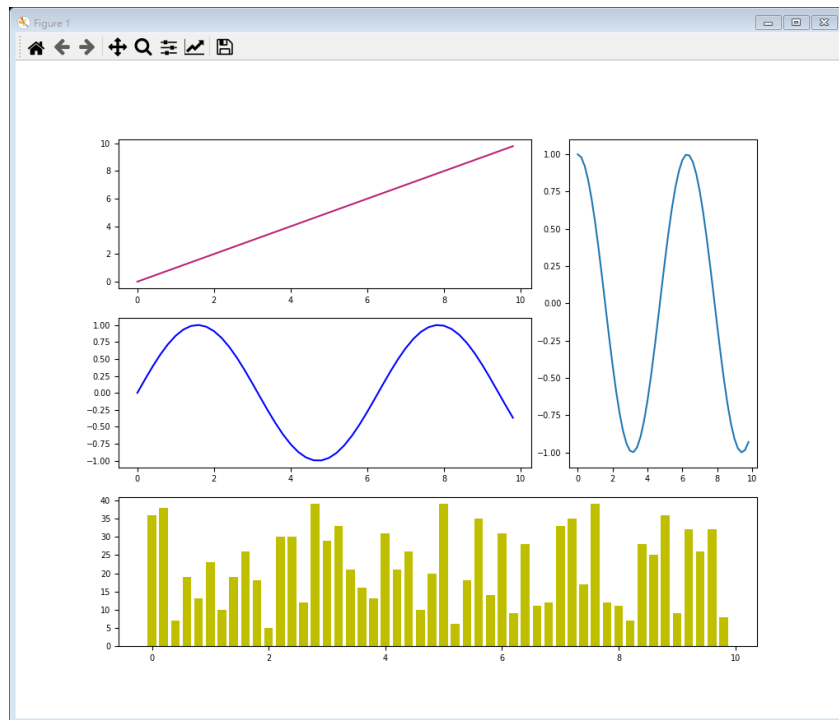
x=np.arange(0,10,0.2)

plt.rcParams['figure.figsize']= [10,8]
plt.rcParams['xtick.labelsize']=7
plt.rcParams['ytick.labelsize']=7

# 隱含宣告產生一個 3 列 x3 行的格柵
ax1= plt.subplot2grid((3,3),
                      (0,0),colspan=2) #從位置 (0,0)，垂直延展兩行
ax2= plt.subplot2grid((3,3), (1,0),colspan=2)
ax3= plt.subplot2grid((3,3), (0,2),rowspan=2) #水平延展兩列
ax4= plt.subplot2grid((3,3), (2,0),colspan=3)

d= np.random.randint(5,40,len(x))
ax1.plot(x, x, color="#bb2277")
ax2.plot(x, np.sin(x), color='b')
ax3.plot(x, np.cos(x))
ax4.bar(x,d , width=0.15, color='y' )
```

執行結果：



另一種方式則是明白地採用 `gridspec` 物件的方式，要使用 `GridSpec` 的功能必須先載入 `gridspec` 模組。`Gridspec` 之功能也就讓我們設定各個子圖佔據之範圍。 以下我們看一個例子：

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

x=np.arange(0,10,0.2)

fig = plt.figure(figsize=(10,8))
params = {'xtick.labelsize': 7,
          'ytick.labelsize': 7 }
plt.rcParams.update(params)

spec = gs.GridSpec(3,3,figure=fig)

#spec[a:b , c:d] 索引用法類似 list 之索引用法
ax1= fig.add_subplot(spec[0,0:2])
ax2= fig.add_subplot(spec[1, :-1])
```

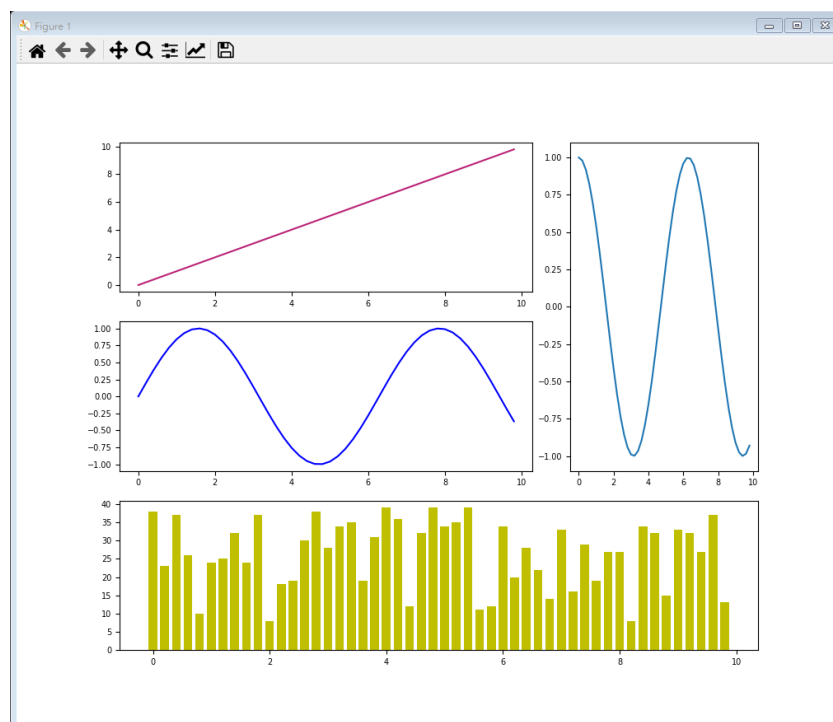
```

ax3= fig.add_subplot(spec[0:2,2])
ax4= fig.add_subplot(spec[2:, :])

d= np.random.randint(5,40,len(x))
ax1.plot(x, x, color="#bb2277")
ax2.plot(x, np.sin(x), color='b')
ax3.plot(x, np.cos(x))
ax4.bar(x,d , width=0.15, color='y' )

```

在上例中我們宣告了一個 3x3 的 gridspec，並透過 figure=fig 設定，來將 gridspec 設定綁住 fig 這個 figure 物件。接著就以用在 add\_subplot() 函式中。 spec 執行結如下：



我們還可以進一步來調整距離跟各行各列之大小比例。

```

import matplotlib.pyplot as plt
import matplotlib.gridspec as gs

x=np.arange(0,10,0.2)
fig = plt.figure(figsize=(10,8))
params = {'xtick.labelsize':7,'ytick.labelsize':7}

```

```

plt.rcParams.update(params)
gs = gspec.GridSpec(3, 3,
                    width_ratios=[1, 1, 2], #設定各列寬度比例
                    height_ratios=[1, 1, 3], #設定各行高度比例
                    figure=fig)

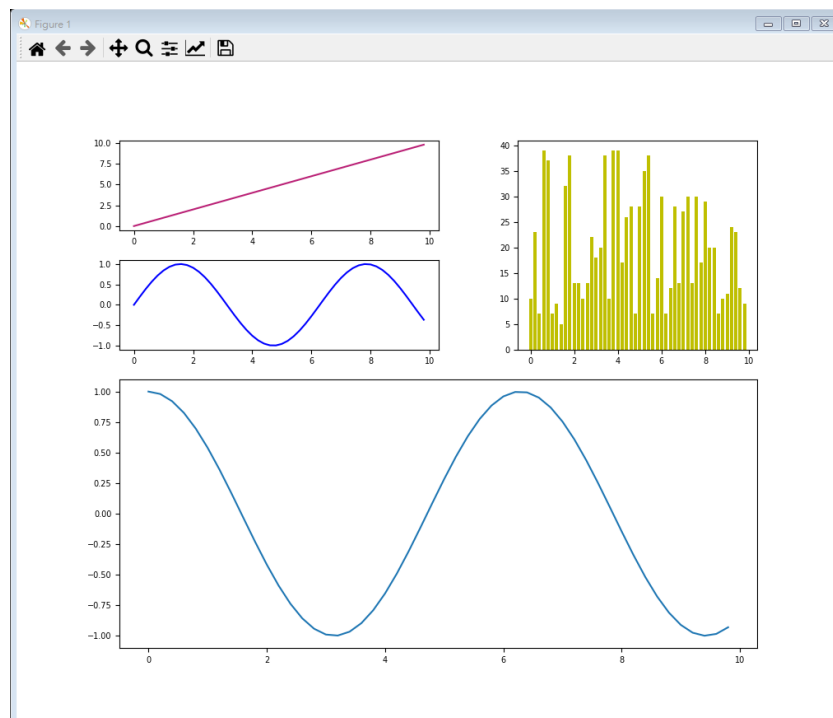
gs.update(wspace=0.5, hspace=0.2) #設定水平與垂直之間距

ax1= fig.add_subplot( gs[0,0:2] )
ax2= fig.add_subplot( gs[1, :-1])
ax3= fig.add_subplot( gs[2:, :] )
ax4= fig.add_subplot( gs[0:2,2] )

d= np.random.randint(5,40,len(x))
ax1.plot(x, x, color="#bb2277")
ax2.plot(x, np.sin(x), color='b')
ax3.plot(x, np.cos(x))
ax4.bar(x,d , width=0.15, color='y' )

```

執行結果：



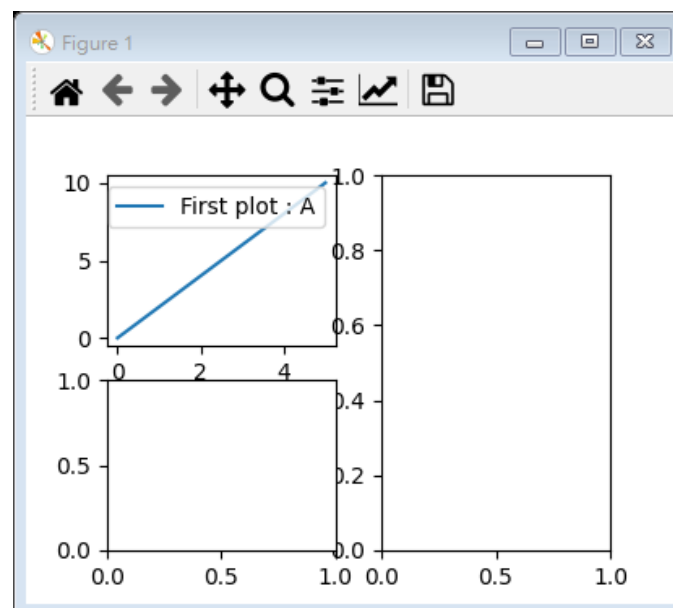
值得一提的一般在排版因為原來圖表的參數設定太大，可能造成各圖的一些標記文字擠在一起的情況。Matplotlib 提供兩種 layout 設定的參數來處理（精確地說是儘可能改善）這個問題。第一種是設定 tight layout，此種設定會自動調整各子圖的參數使得所有子圖能夠適當地塞進整個 figure 的範圍，當然此種調整不一定能百分之百 OK，還是要看整個圖的大小限制。而且此種設定只會檢查標題和一些標記（ticklabels, axis labels, title...）的範圍。以下是一個例子：

```
import matplotlib.pyplot as plt

plt.figure(figsize=(4,4))
ax1= plt.subplot2grid((2,2),(0,0) )
ax2= plt.subplot2grid((2,2),(0,1), rowspan=2)
ax3= plt.subplot2grid((2,2),(1,0) )

ax1.plot( range(6),range(0,12,2),label="First plot : A")
ax1.legend()
```

執行結果：



現在加入 `tight_layout()` 命令：

```
import matplotlib.pyplot as plt

plt.figure(figsize=(4,4))
```

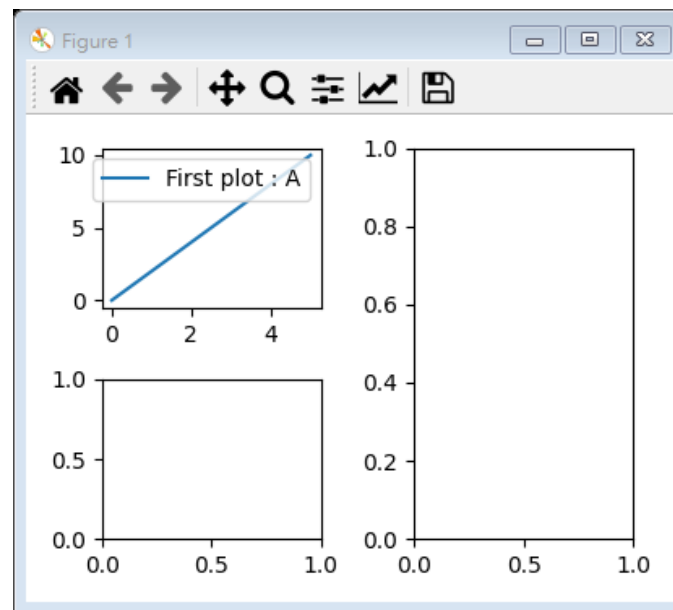
```

ax1= plt.subplot2grid((2,2),(0,0) )
ax2= plt.subplot2grid((2,2),(0,1), rowspan=2)
ax3= plt.subplot2grid((2,2),(1,0) )

ax1.plot( range(6),range(0,12,2),label="First plot : A")
ax1.legend()
plt.tight_layout()

```

執行結果：



可以看出來並非考量第一子圖之 `legend()` 長度。

另一種方式是宣告 `constrained_layout` 參數值為 `True`，請看下面例子：

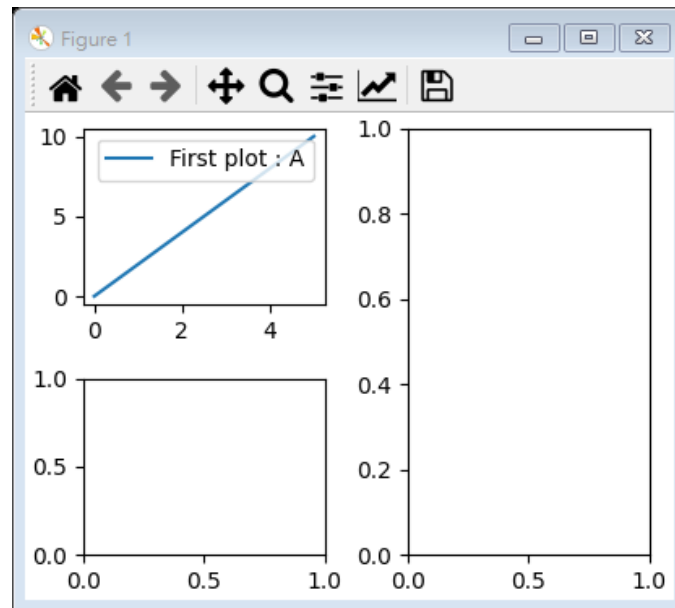
```

import matplotlib.pyplot as plt
plt.figure(figsize=(4,3), constrained_layout=True)

ax1= plt.subplot2grid((2,2),(0,0) )
ax2= plt.subplot2grid((2,2),(0,1), rowspan=2)
ax3= plt.subplot2grid((2,2),(1,0) )
ax1.plot( range(6),range(0,12,2),label="First plot : A")
ax1.legend()

```

此時執行之結果：



可以看得出來考量之因素較多，不是只有看 `tick_labels`。