Lesson 1 函數的繪製

林哲兆

January 9, 2024

本文使用 Python 的 Matplotlib 來繪製函數，除了基本的函數外，還包括三角函數、Sigmoid 函數以及常態分配 PDF 等，並會在最後一小節討論一個小型專題。另外，本文除了將函數繪製完成之外，也會特別將需要注意的程式碼列出來，用以紀錄避免未來發生同樣的錯誤。

1 函數繪圖

1.1 三角函數

第一題包括四個函數，其中第四個是自行添加的，目的都是希望利用圖片能找出函數在 x 趨近於 0 時的極限，使用四個函數如下，都是經典的三角函數極限題目，使用羅必達可以迅速解出，但我們想利用函數的繪製來觀察 x 趨近於 0 時的極限。以下為四個函數與對應圖形：

$$y = \frac{\sin x}{x}$$
$$y = \frac{\sin x^2}{x}$$
$$y = \frac{(\sin 2x)^2}{x^2}$$
$$y = \frac{1 - \cos x}{x^2}$$

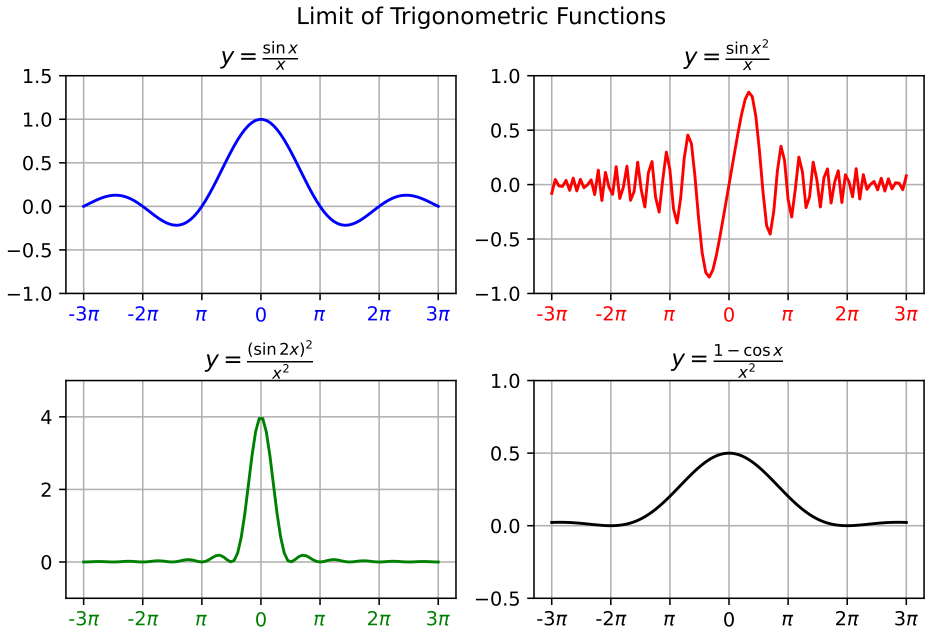


圖 1: 使用 subplot 展示四個函數圖形

藉由圖 1 我們可以看出四個函數的極限分別為:

$$\lim_{x \to 0} \frac{\sin x}{x} = 1$$

$$\lim_{x \to 0} \frac{\sin x^2}{x} = 0$$

$$\lim_{x \to 0} \frac{(\sin 2x)^2}{x^2} = 4$$

$$\lim_{x \to 0} \frac{1 - \cos x}{x^2} = \frac{1}{2}$$

在繪製這題的時候，我們需要特別注意的有幾項:

- 透過 subplot 產生 $2/times2$ 的子圖之後，若要在某一格繪圖，必須用括號去指定要在哪張圖作畫，參數是 m、n，也就是第 m 列、第 n 行 (m,n 從 0 開始)。
- 在設定 ax 變數後，要改變 x、y 軸的上下限、下標，需要加上 set。
- 在標題的部分子標題的語法與主標題有所不同。
- 如果要調整子圖之間的距離，可以用 vspace 或 hspace 調整間距。

範例程式碼如下：

```
ax[0, 0].plot(x, y, color='b')              #ax.[m, n]
ax[0, 0].set_ylim([-0.5, 1])                #ax需要加set_
ax[0, 0].set_xstick([...])
ax[0, 0].set_title('.........')             #子標題
plt.suptitle('.........')                   #主標題
plt.subplots_adjust(hspace=0.4)             #調整距離
```

1.2 sigmoid 函數

第二題是在機器學習會看到的 sigmoid 函數，也叫做羅吉斯函數，在神經網絡中它經常被當作激活函數，尤其是在早期的神經網絡中。這題我們將畫出它的圖形，並透過更改不同的 a 值，來觀察圖形的走向是如何，以及這個函數的漸進線。

$$y = \frac{e^{\alpha x}}{1 + e^{\alpha x}}$$

此即 sigmoid 函數，圖形呈現 s 形，由圖可以看出這個函數 y 值會介於 01 的，也正因為如此，最常看到的應用情境，就是當我們在訓練模型做二分類的時候。或是我們將模型的最後一層神經網路設定為只有一個神經元，再將最後神經元所輸出的值輸入 sigmoid 函數，這樣一來，就會得到一個介於 $[0, 1]$ 之間的數值，即可進行分類。由於函數會介於 0 到 1 之間，因此兩條漸進線分別是 $y = 0$ 與 $y = 1$，當 x 趨近無限大時 y 會靠近 1; 當 x 趨近無限大時 y 會靠近 0, 圖形與漸進線的繪圖如下：

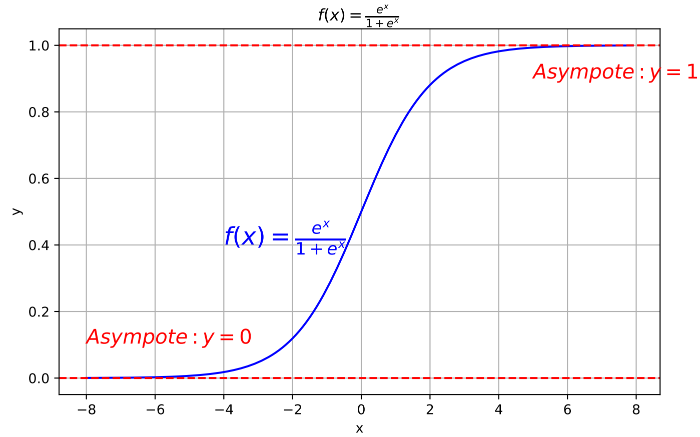


圖 2: $\alpha = 1$ 時的 Sigmoid 函數

圖 2是當 $\alpha = 1$ 時的結果，如果將 $\alpha = 2, 3, 4, 5$ 的圖形一起畫出來，如圖 3 ，可以發現圖形隨著 α 值增加，函數會更加陡峭，也就是在 $x = 0$ 之間的變動會非常大。

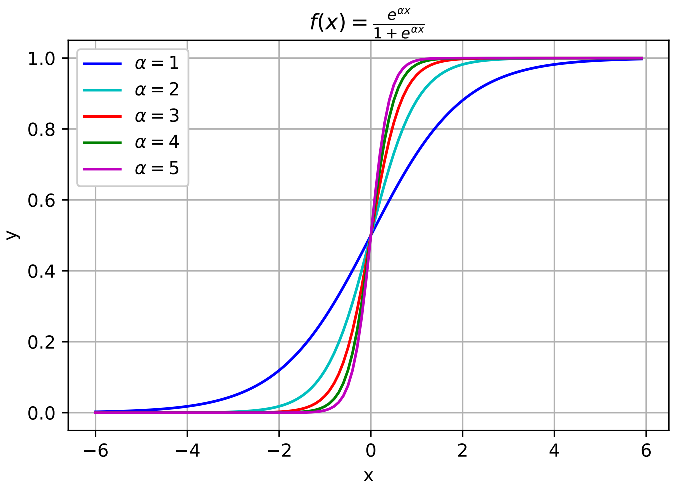


圖 3: 不同 α 值的 Sigmoid 函數

圖 3 是使用迴圈畫的，其中有幾個地方需要注意：

- 顏色要自己選取會比較麻煩，除非再寫一個隨機生成顏色的函數。
- 可以把標籤也用迴圈的參數表示，但前面須加上一個小 f，這樣使用 plt.legend() 就會產生五條線分別的標籤。

以下是附上迴圈的寫法：

```
    for alpha in range(1,n+1):
    y = np.exp(alpha * x) / ( 1 + np.exp(alpha * x))
```

```
    plt.plot(x, y, label = f'$\\alpha={alpha}$', color =
        colors[alpha-1])
```

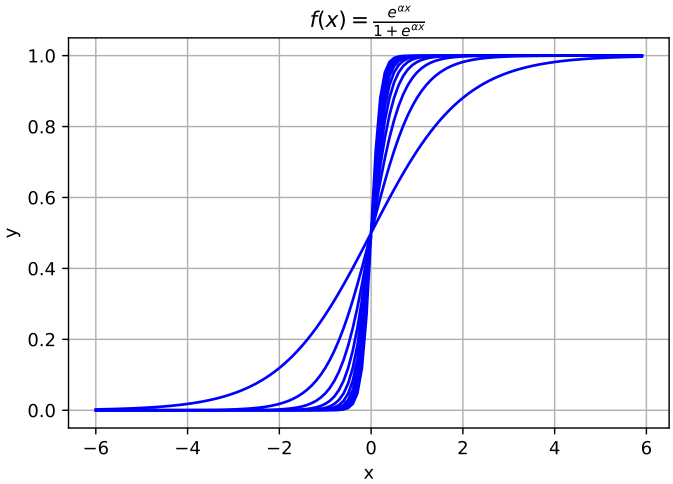


圖 4: 更多不同 α 值的 Sigmoid 函數

圖 4 是使用 broadcast 畫的，相較迴圈可以畫更多不同 α 值，運算速度也較快，但顏色要改變相對就比較麻煩，如果只是需要看出趨勢，使用 broadcast 來畫圖會更快且程式碼更精簡。片段程式碼如下：

```
    alpha = np.arange(1, n+1 , 1)
    y = np.exp( x.reshape(-1 , 1) * alpha) / ( 1 +
              np.exp(x.reshape(-1 , 1) * alpha))
    plt.plot(x, y, color = 'b', alpha = 0.5)
```

1.3 震盪函數

第三題我們要繪製的函數如下：

$$y = e^{-\frac{x}{10}} \sin(x)$$

這是一個指數函數與三角函數的結合函數，由於 $e^{-\frac{x}{10}}$ 隨著 x 變大值會變小，因此使得後面的 $\sin(x)$ 震盪幅度也隨著 x 值變大而變小，從而讓整個函數產生一個震盪衰減的趨勢，適合呈現這個特徵的 x 軸範圍大約介於-40 到 10 之間，函數在 x 超過 10 之後震盪幅度接近 0，圖形如下所示：

繪製此函數的時候步驟幾乎與前面相同，決定適合的 x 值域就好，但在呈現畫趨勢的兩條灰線時，有使用到別的語法，我使用的是 scipy 裡的 make_ interp_ spline，透過這個函數可以自選幾個座標點，讓它們連起來會是一條曲線，而不

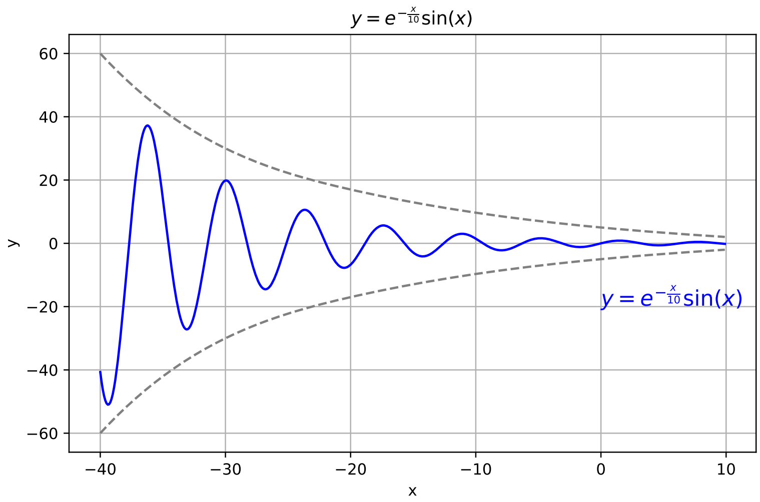


圖 5: 函數 $y = e^{-\frac{x}{10}} \sin(x)$ 的走勢

是直線，座標點需要多次測試，才能找到完全能符合震盪趨勢的曲線。曲線語法如下：

```
from scipy.interpolate import make_interp_spline
x = np.arange(-40, 10, 0.1)
a = np.array([-40, -30, -20, 0, 10])
b = np.array([-60, -30, -17, -5,-2])
curve1 = make_interp_spline(a, b)
ys=curve1(x)
plt.plot(x, ys, linestyle=’--’,color=’gray’)
```

1.4 定義域非實數的函數

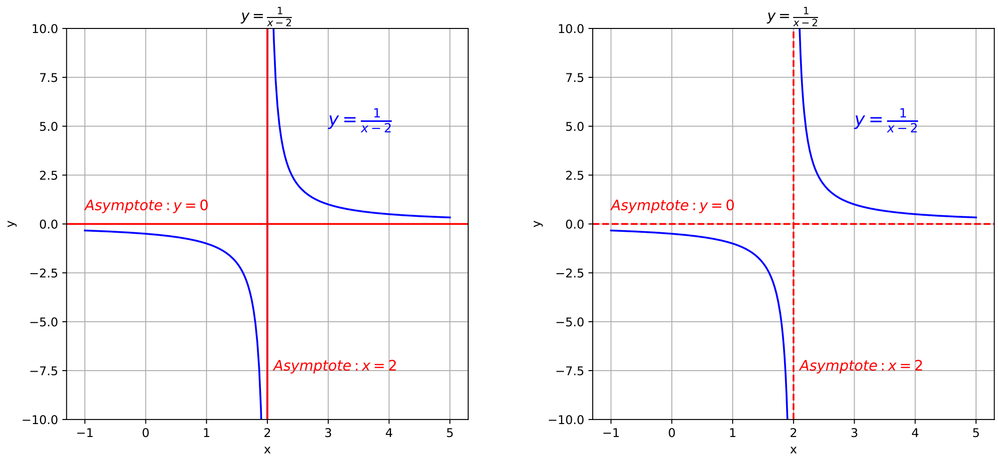
第四題要繪製的函數如下：

$$y = \frac{1}{x-2}$$

這個函數在 0 的時候沒有定義，所以我們在設定 x 的值域時，須把 2 排除在外，除了可以分成 -2 左邊一段與 -2 右邊一段外，也可以使用集合的概念，設定一段 x 的區間，並計算這個區間與 2 的差集，就能產生不包含 2 的一段區間。語法如下：

```
x = np.setdiff1d([np.linspace(-1, 5, 200)],[2])
```

np.setdiff1d 是差集的函數，透過這個函數我們可以產生正確的 x 的 domain，然後畫出下圖左邊的 (a) 圖：



(a) 未更改值域的圖形

(b) 更改值域後的圖形

圖 6: $y = \frac{1}{x-2}$ 與其漸進線

值得注意的是這樣做 plot 函數會把 y 值趨近於 ∞ 與 $-\infty$ 的兩點連起來，但實際上的圖形是不會連起來的，如果要解決的話似乎只能做兩次圖，一次是 $x > 2$ 的圖，一次是 $x < 2$ 的圖，最後畫出如圖 6 右邊的圖。

1.5 多項式函數與反函數

第五題要繪製的函數為：

$$y = x^3 + 2$$

除此之外要加上這個函數的反函數，而我們可以知道一個函數與它的反函數會對稱於 $y = x$ 這條斜直線，因此我們在繪製完兩個函數之後會再加上他們的對稱線與交點，交點透過解方程式可以算出來，兩函數交於 $(-1.54, -1.54)$，最後呈現的圖形如下，加上對稱線後像一隻三叉戟：

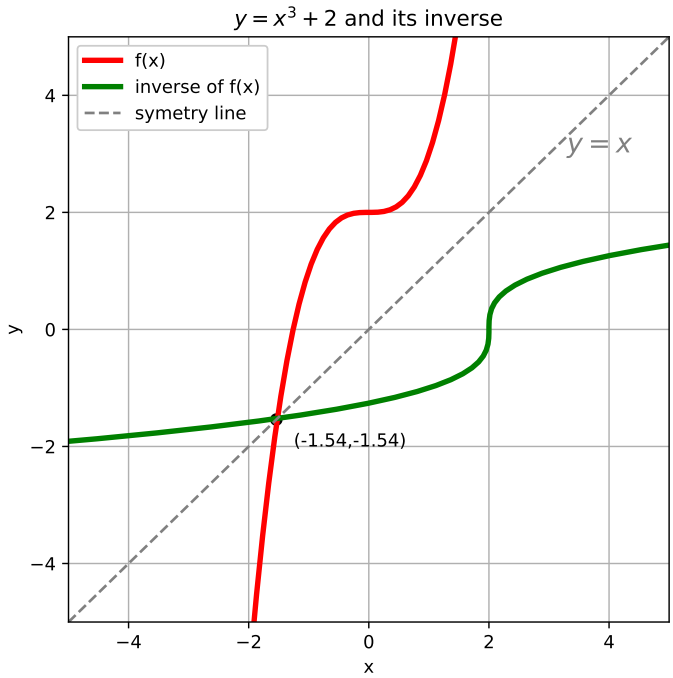


圖 7: 函數 $y = x^3 + 2$ 與它的反函數

這題為了清楚的呈現兩者對稱的圖形，需要把 x 的值域設置的窄一點，因為三次方程式的函數值膨脹的非常快，設置太寬可能看不出變化。另外，繪製反函數非常容易，只需要把 x 與 y 順序對調就能輕鬆畫出反函數，不需要特地去算反函數。程式碼如下：

```
plt.plot(x , y , label = 'f(x)', color = 'r',linewidth
    =3)
plt.plot(y , x , color = 'g', label = 'inverse of f(x)
    ',  linewidth=3)
plt.scatter(-1.54,-1.54,color='black')
```

在進行描點的時候要用到的語法是 plt.scatter，裡面有 marker、color、size 等參數可以控制點的形狀、大小與顏色，marker 默認的點會是實心點，所以這題不更改 marker 的參數，至於 size 也覺得差不多，所以就不做更改。

1.6 常態分配的機率密度函數

第六題要呈現的函數是常態分配的機率密度函數：

$$y = \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-3)^2}{2}}$$

由函數可以明顯看出是一個 $\mu = 3$，$\sigma^2 = 2$ 的常態分配，因此在決定 x 軸範圍的時候要把 $x = 3$ 放在圖的正中央，以呈現常態分配對稱的情形。另外，我更改 x 軸的座標，讓 x 的座標是以距離平均數幾個標準差的形式呈現，因為使用常態分配不外乎就是檢定某筆資料是否距離平均數很遠，透過把 x 的座標更改，可以更清楚呈現某筆資料是否距離平均數很遠，以及距離幾個標準差。最後圖形呈現如下：

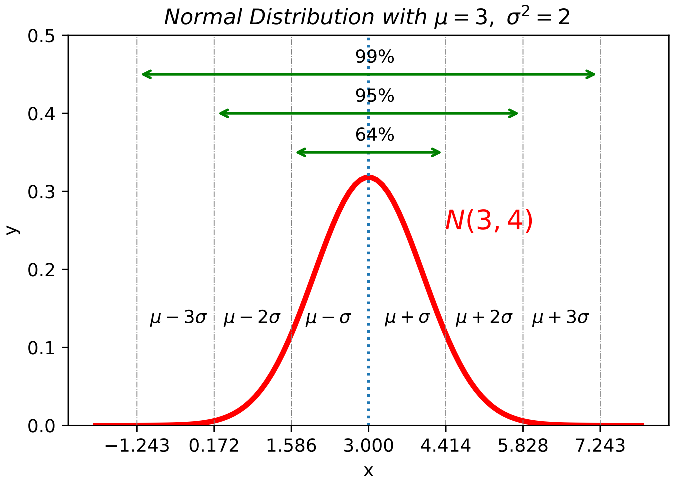


圖 8: 函數 $y = \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-3)^2}{2}}$

這題在繪出主圖形時與前面過程大致相同，但在繪製區隔線的時候，我希望一次全部畫出來，於是使用迴圈畫這六條鉛直線，過程中我原本想要更改直線的顏色，需要讓迴圈跑兩個變數，所以用了 zip 的語法，讓 x 值與顏色值可以一起同時變動，但最後礙於顏色太多很雜亂還是統一灰色，下面是迴圈程式碼：

```
x_values = [avg-3*sigma, avg-2*sigma, avg-1*sigma,
   avg+sigma, avg+2*sigma, avg+3*sigma]
colors = ['gray', 'gray', 'gray','gray', 'gray', 'gray
   ' ]
for x, color in zip(x_values, colors):
    plt.axvline(x=x, color=color,    linestyle='-.',
       linewidth=0.5)
```

繪製完成圖形後為了讓讀者了解這幾條線的意義，於是使用 plt.text 在圖上標註是距離平均數幾個標準差，最後再把 N(3,4) 加上就完成繪製了。

1.7 第七題

這題的目標是繪製一個簡單的多項式函數，並找出這個函數的最大值與它的根，函數如下所示：

$$y = 3x^3 - x^4$$

這題可以透過一階微分計算找出最大值的位置，而 $y = f(x)$ 的根可以也透過簡單的數學解出來，同時兩者要用 python 的程式碼去解出來也是行得通的。下面是解一階微分程式碼：

```
import sympy as sp
X = sp.symbols('X')
F = 3*X**3-X**4
df = sp.diff(F, X)
solutions = sp.solve(df, X)
for sol in solutions:
    print(f'X = {sol}, F\'(X) = 0')
```

透過 Sympy 套件，設定好原函數以及微分的對象之後，塞到 sp.diff 裡就會產生一階導函數，再將導函數拿去 solve，就能計算出解。而由於解不只一個，因此用迴圈把結果 print 出來。

得到結果後一樣使用 plt.scatter 來把點描上去，但特別的是因為要區別最大值與根的點，所以把兩個不同的點使用 label 做標籤，最後加上 legend 後就可以得到下圖左方的結果：

```
fig,ax = plt.subplots(1, 2, figsize=(12, 4))
ax[0].plot(x, f(x), color='b')
ax[0].scatter(x_roots,y_roots, color='red', marker='o
   ', label='roots')
ax[0].scatter(x_max, y_max, color='black', marker='o',
    label='maximum')
```

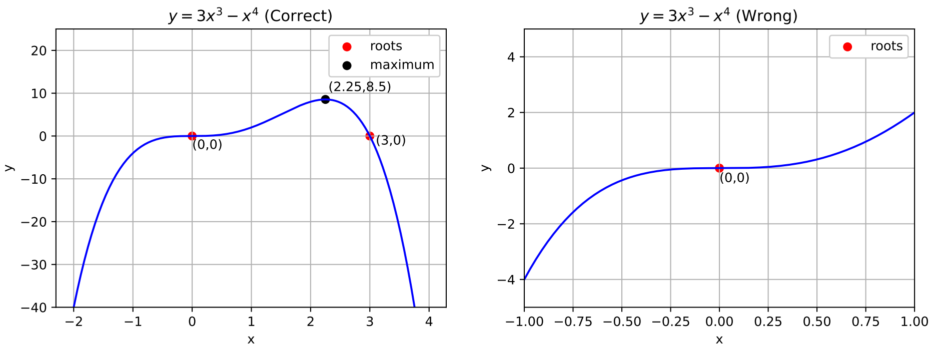


圖 9: 函數 $y = 3x^3 - x^4$，正確與錯誤示範

圖 9 的右圖是一個錯誤示範，因為沒有設定好 x 的值域，所以圖最後只看得出一個解，為避免此種情況發生最好多嘗試幾種 x 的值域，等大致上確定圖形趨勢後，再選擇一個相對適合的值域，否則無法完整呈現函數圖形。

1.8 常見極限函數

這題的函數常被用來作為極限的題目，用羅必達法則可以很快的解出來，函數如下所示：

$$y = \frac{\ln x}{x^2}$$

透過羅必達法則我們可解出：

$$\lim_{x \to 0} \frac{\ln x}{x^2} = -\infty$$

$$\lim_{x \to \infty} \frac{\ln x}{x^2} = 0$$

所以我們可以知道這個函數會存在著兩條漸進線，分別是 $x = 0$ 與 $y = 0$，將漸進線與函數圖形一同繪出後，我們還需計算出最大值的位置，透過上一節 Scipy 的套件可以計算出來，最後標上最大值的點就能產生下圖：

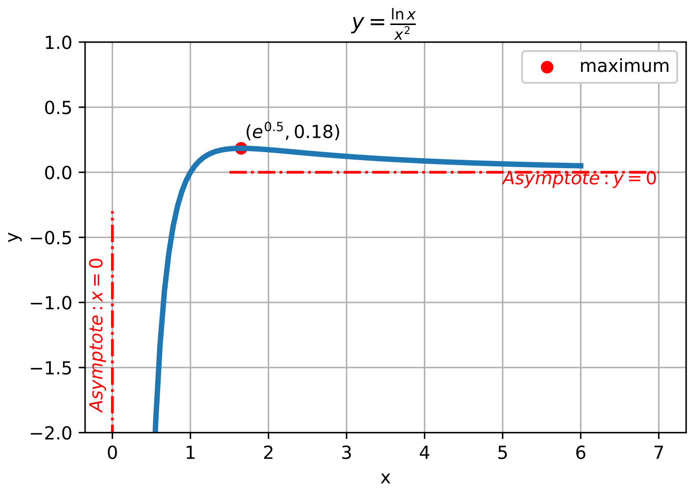


圖 10: 函數 $y = \frac{\ln x}{x^2}$ 與其漸進線

這個圖形在設定 x 的值域的時候要當心，起始點不能離 0 太近，一開始我設定的範圍在 0.01 到 7 之間，但畫出來後發現，0.01 會使得 y 值過大，導致在標 text 的時候無法標到正確的位置上，於是最後更改到 0.7 到 7，讓函數圖形可以在保有其趨勢的情況下，還能正確顯示。另外，在 plt.text 的參數中，有一個 rotation 可以調整，指定數字就可以讓文字旋轉指定的度數，例如 rotation=90，就是讓

文字由水平變成垂直，其預設是 0，所以文字都會以水平顯示，語法如下與部分繪製的程式碼如下：

```
fig ,ax = plt.subplots()
ax.plot(x , f(x) , linestyle='-',linewidth=3)
ax.hlines(y=0 ,xmin=1.5, xmax=7,linestyles='-.' ,
   colors='r')
ax.vlines(x=0 ,ymin=-3, ymax=-0.3,linestyles='-.' ,
   colors='r')
ax.set_ylim([-2,1])
ax.scatter(x_max, y_max, color='r',label='maximum')
```

1.9 水平分段函數

第九題是一個分段函數，它的值域與對應函數值如下：

$$f(x) = \begin{cases} 1, & 1 \leq x < 3 \\ 2, & 3 \leq x < 5 \\ 3, & 5 \leq x < 7 \end{cases}$$

這題的繪圖相對來說簡單，需要的注意是有等號的一邊需要加上實心點，沒有等號的部份加上空心點，其中控制空心與實心的參數是 facecolors，當 facecolors=none，就會畫出一個空心的點。以下是部分繪圖的程式碼與繪製完成的圖形：

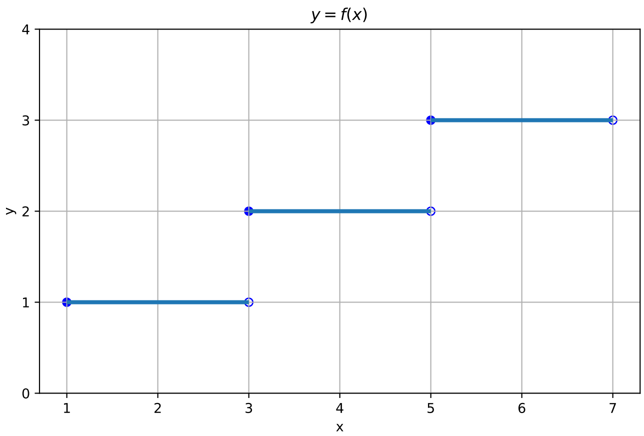


圖 11: 函數 $y = f(x)$

```
ax.hlines(y=1 ,xmin=1 ,xmax=3,linewidth=3)
ax.hlines(y=2 ,xmin=3 ,xmax=5,linewidth=3)
ax.hlines(y=3 ,xmin=5 ,xmax=7,linewidth=3)
ax.scatter(x1_point ,y1_point ,marker='o',color='blue
   ')
```

```
ax.scatter(x2_point ,y2_point ,marker='o',color='blue
   ', facecolors='none')
```

1.10 圓形

第十題要畫的函數是一個半徑為 1，圓心在 $(0,0)$ 的圓：

$$x^2 + y^2 = 1$$

畫圓有很多方法，首先介紹第一個，使用極座標表示：

先把 θ 範圍定義出來，再將 x 設為 $\sin(\theta)$，y 設為 $\cos(\theta)$，這時候將 x,y 放入 plot 函數就能畫出一個圓。以下為部分程式碼與繪製的圖：

```
theda = np.linspace(-2*np.pi , 2*np.pi,100)
x = np.sin(theda)
y = np.cos(theda)
fig, ax = plt.subplots()
plt.plot(x,y)
```

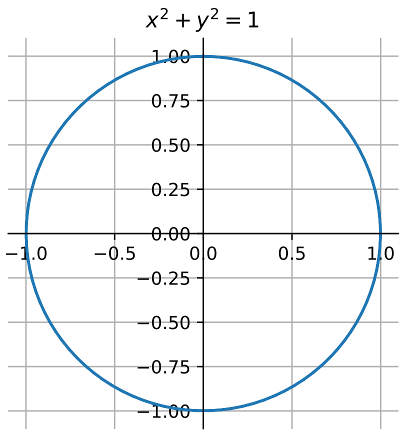


圖 12: 函數 $x^2 + y^2 = 1$

第二個方法可以使用 matplotlib 內建的函數，也就是 matplotlib.patches，透過這個套件可以直接指定半徑與圓心，不需要設定值域就能將圓畫出來，也可以指定要不要填滿圓以及圓的顏色，而最後的繪圖結果會與圖 12 相同，部分程式碼如下：

```
import matplotlib.patches as ptc
circle = ptc.Circle((0, 0), radius=1, fill=False,
   color='blue')
fig, ax = plt.subplots()
ax.add_patch(circle)
```

第三個方法使用的是隱函數的方法，也就是將方程式表示成:

$$x^2 + y^2 - 1 = 0$$

先將 x、y 分別設置為兩個一維值域，再透過 numpy 裡的 meshgrid 讓 x,y 由分別的一維數據轉換成結合的二維數據，最後透過等高線的函數 plt.contour，把符合這個隱函數的點繪製上去，連成等高線，就能繪製出圓。

其中在 contour 中有一個 level 參數可以控制隱函數的值，也就是讓隱函數等於多少，這題我們希望隱函數 $x^2 + y^2 - 1 = 0$，所以將 level 設置為 0，於是最後也可以畫出與圖 12 相同的圖，部分程式碼如下：

```
x = np.linspace(-1, 1, 200)
y = np.linspace(-1, 1, 200)
x, y = np.meshgrid(x, y)
circle = x**2 + y**2 - 1**2
plt.contour(x, y, circle, levels=[0], colors='blue')
```

繪製出圓後，為了要符合老師講義的座標軸，以及把座標軸設置為正方形，我們還需要加上幾行程式碼：

```
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_position(('data', 0))
ax.spines['bottom'].set_position(('data', 0))
plt.axis('equal')
```

ax.spines 可以把上、下、左、右的邊界進行位移或隱藏，位移使用 set_ position，隱藏則使用 set_ visible，最後一行則是將座標軸設置為等寬，透過這幾行程式碼，就能將圖片調整成與老師講義相同的格式。

1.11 正方形

第十一題要繪製的是一個正方形，正方形一樣有不少方法可以畫出來，首先介紹第一個，也是用 matplotlib.patches 直接畫出來，只要指定左下的頂點與長、寬，就能產生一個正方形，再把座標軸調整一下就能畫出與講義相同的圖形，程式碼與圖形如下：

```
fig, ax = plt.subplots()
square = ptc.Rectangle((-0.5, -0.5), 1, 1, linewidth
   =2,   edgecolor='blue', facecolor='none')
ax.add_patch(square)
```

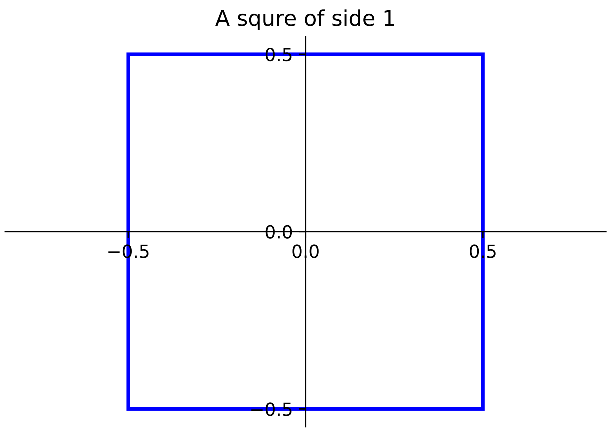


圖 13: 邊長為 1 正方形

第二種則是把頂點座標挑出來，形成一個陣列，再讓 plot 函數把他們連起來，值得注意的是起始點與終止點必須一樣，這樣就能畫出與圖 13 一樣的圖，部分程式碼如下：

```
x = [-0.5, 0.5, 0.5, -0.5, -0.5]
y = [-0.5, -0.5, 0.5, 0.5, -0.5]
plt.plot(x, y, color='blue')
```

最後一種是用 hline 與 vline 畫出兩條垂直線與兩條水平線，讓四條線為成一個正方形，部分程式碼如下：

```
plt.vlines(x=[-0.5, 0.5], ymin=-0.5, ymax=0.5, color='
   blue')
plt.hlines(y=[-0.5, 0.5], xmin=-0.5, xmax=0.5, color='
   blue')
```

2 專題

令 $S_n = \sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{k}$

1. 驗證 $\lim_{n\to\infty} S_n$ 發散
2. 令 γ_n 為圖 14 的陰影處面積。驗證 $\gamma_n = S_n - \ln(n+1)$（可以直接證明）
3. 驗證 $\frac{1}{2}(1 - \frac{1}{n+1}) < \gamma_n < 1$

2.1 第一小題

雖然微積分教過我們 S_n 這個級數和會發散，但我們在紙上始終無法算出當 n 很大的時候，S_n 到底是多少，但借用電腦我們可以找出來，利用單迴圈我們可以計算 S_n 的值，外面再加一層迴圈，可以觀察隨著 n 越來越大，S_n 的情況，圖形與程式碼如下：

```
ns = [1, 10, 100, 1000,10**5,10**6] ##加到多少
result = []              ##預先留空間
for n in ns:                     ##把指定的n都做過
    Sn = 0
   Sn_values = []
    for k in range(1, n + 1):    ##級數計算
        Sn = Sn+ 1 / k
        Sn_values.append(Sn)     ##把舊值覆蓋
    result.append(Sn_values)     ##把結果放進[]

  plt.figure(figsize=(8, 5))
for i in range(len(ns)):         ##總共畫6次
  ##把每次x,y的點都用i來表示寫出迴圈
    plt.plot(range(1, ns[i] + 1), result[i])

##取指定的n與對應的Sn出來，點在圖上
for i, n in enumerate(ns):
  ##result取每次畫圖出來的最後一項，當作縱軸值
    plt.scatter(n, result[i][-1], c='red')
    ##標籤n={n}要放在比指定點高10單位的地方,並置中
    plt.annotate(f'n={n}', (n, result[i][-1]),
       textcoords="offset points", xytext=(0,10), ha='
       center')
```

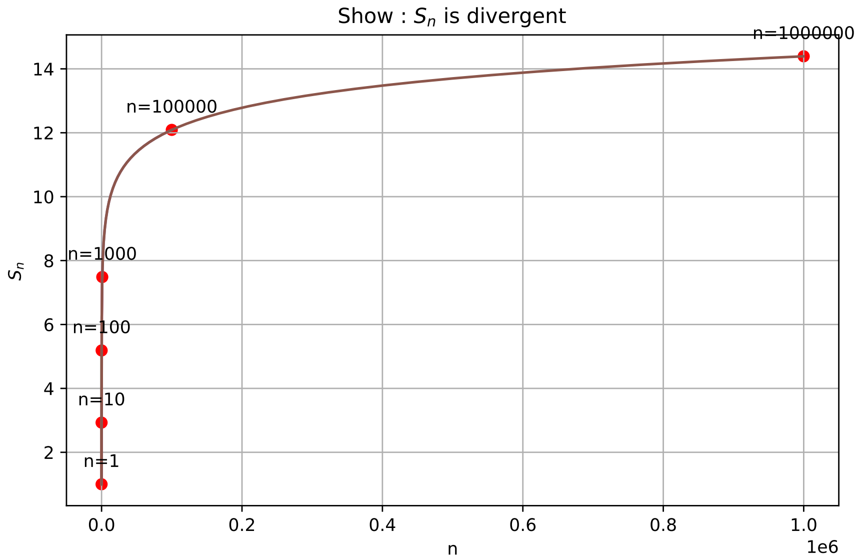


圖 14: S_n 隨著 n 的變化

從圖中我們可以看出雖然 S_n 值增加的速度有趨緩，但始終找不到一條漸進線可以趨近，即便 n 到了 1000000，還是有在慢慢增加的趨勢，因此可以知道這個級數是發散的。

2.2 第二小題

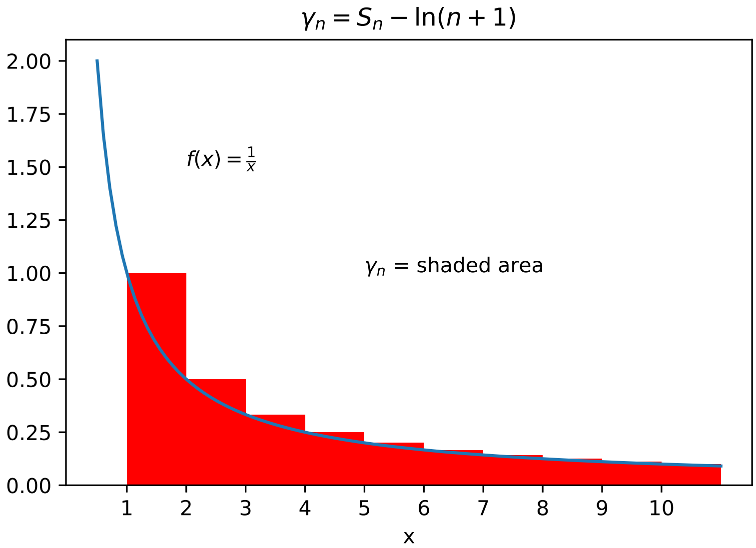


圖 15: γ_n = sum of the red areas

第二題由圖 15 可知，γ_n 的值是所有長條面積加起來減掉 $y=\frac{1}{x}$ 從 1 積到 $n+1$ 的積分值，其中長條圖面積為：

$$1\times 1+1\times\frac{1}{2}+1\times\frac{1}{3}+1\times\frac{1}{4}+\cdots+1\times\frac{1}{n}=S_n$$

$y=1/x$ 從 1 積到 $n+1$ 的積分值積分值為：

$$\int_1^{n+1}\frac{1}{x}dx=\ln(n+1)-\ln(1)=\ln(n+1)$$

因此可得證：

$$\gamma_n=S_n-\ln(n+1)$$

在畫圖 15時，要先把長條圖與函數 $y=\frac{1}{x}$ 先畫好，再來要蓋掉曲線下的長條圖，也就是把 $y=\frac{1}{x}$ 與 x 軸圍出的面積填上白色，使用 fillbetween 語法就能做到，繪圖程式碼如下所示：

```
a, b, n = 1, 10, 10
x1 = np.linspace(a, b, n)
x2 = np.linspace(0.5, 11, 100)
y1 = 1 / x1
y2 = 1 / x2

plt.bar(x1, y1, 1, align='edge', color='red', alpha
   =0.5)
plt.plot(x2, y2)
plt.fill_between(x2, y2, 0 , where =y2 > 0, color = '
   white')
plt.xlabel('x')
plt.xticks(range(1,11,1))
plt.title('$\gamma_n=S_n-\ln (n+1)$')
plt.text(2, 1.5,'$f(x)=\\frac{1}{x}$')
plt.text(5,1, '$\gamma_n$ = shaded area')
plt.savefig("pj2.eps", format="eps", dpi=300)
plt.show()
```

2.3 第三小題

第三小題使用圖形比較，分別繪製三條函數的圖形：

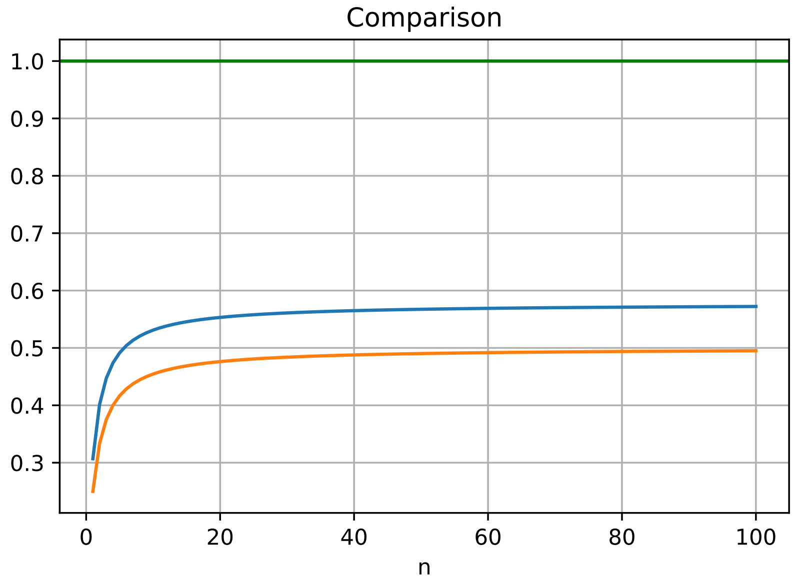


圖 16: $\frac{1}{2}(1-\frac{1}{n+1}) < \gamma_n < 1$

由圖片可以清楚看出 $\frac{1}{2}(1-\frac{1}{n+1}) < \gamma_n < 1$。這題的 S_n 改用第二種方法，使用 cumsum 來計算 S_n，也可以算出級數和，另一方面這樣呈現 γ_n 也比較直觀，附上繪圖程式碼：

```
n= np.arange(1, 101)
S_n = np.cumsum(1 / n)
gamma_n = S_n - np.log(n + 1)
y_n = 1/2*(1-1/(n+1))

plt.plot(n, gamma_n, label='$\gamma_n = S_n - \ln(n+1)
   $')
plt.plot(n, y_n, label='$1/2*(1-1/(n+1))$')
plt.axhline(1, xmin=0, xmax=100, label='1',color='
   green')

plt.xlabel('n')
plt.title('Comparison')
plt.grid(True)
plt.savefig("pj3.eps", format="eps", dpi=300)
plt.legend()
plt.show()
```

3 結論

本文透過繪製函數圖形，介紹了許多 python 的基礎語法，包括 numpy、matplotlib 套件等，但在繪製圖形時，不只要確保程式碼順利執行，也要讓讀者清楚看見圖形所要表達的主旨，所以註解與圖形範圍選擇也是很重要的一部分，希望藉由本文，讀者對使用 python 繪圖能有更深的了解。