計算機概論 Final Project
泰勒級數逼近之數值分析與視覺化

姓名： 周明坤

科系： 不分系

學號： AN4096750

GitHub： `https://github.com/MKChou/Taylor-Series-Aproximation`

目录

1 問題定義與理解 (Problem Definition & Understanding)

電腦其實沒辦法直接計算 $\sin(x)$、$\cos(x)$ 或 e^x 這些複雜的函數。所以我們需要用**多項式近似（Polynomial Approximation）**的方法來計算出這些函數的值。

泰勒級數（Taylor Series）就是把一個函數拆解成很多項多項式加起來。這次專案我選了 $\sin(x)$ 來研究，用 Python 寫程式看看不同項數的泰勒級數能多準確地逼近真實的 $\sin(x)$，還有誤差會怎麼變化。

1.1 為什麼電腦需要逼近法？

1. **硬體限制：**電腦只會做加減乘除，沒辦法直接計算 $\sin(x)$ 這種函數。
2. **效率考量：**如果每個函數都要用查表或硬體來做，成本太高了。用逼近法可以根據需要的精度來決定要算幾項，使用起來比較彈性。
3. **實用性：**透過調整項數可以在準確度和計算時間之間找到平衡。

2 數學背景 (Mathematical Model)

$\sin(x)$ 在 $x = 0$ 處的麥克勞林級數（Maclaurin Series，泰勒級數的一種特例）公式如下：

$$\sin(x) = \sum_{i=0}^{\infty} \frac{(-1)^i}{(2i+1)!} x^{2i+1} \tag{1}$$

展開後為：

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots \tag{2}$$

實際上我們不可能計算無限項，所以只能取前 n 項來近似。這次專案我試了不同的 n 值，看看效果如何。

通項公式：

$$T_i = \frac{(-1)^i}{(2i+1)!} x^{2i+1} \tag{3}$$

其中：

- i 為項數 index（從 0 開始）
- $(2i+1)!$ 為階乘
- $(-1)^i$ 提供交替的正負號

2.1 收斂性分析

- **收斂半徑：** $\sin(x)$ 的麥克勞林級數對所有實數 x 都收斂（收斂半徑為無窮大）。
- **誤差特性：**當 $|x|$ 較小時，級數收斂較快；當 $|x|$ 較大時，需要更多項才能達到相同精度。

3 程式實作與效率 (Programming & Efficiency)

這次專案我用 Python 來寫，主要用了幾個工具：

1. **NumPy 的向量化運算：**不用 Python 的 `for` 迴圈一個一個算，而是用 NumPy 陣列一次處理整個範圍的 x 值。我設定了 x 從 -2π 到 2π，取 1000 個點，用向量化運算可以一次處理所有點，這樣會快很多。
2. **SciPy 的階乘函數：**用 `scipy.special.factorial` 來算階乘，這樣即使項數很多也不會出錯。例如當 $n = 8$ 時，需要計算到 $15!$，這個數字很大，用 SciPy 的函數可以確保數值穩定。
3. **Matplotlib 畫圖：**用 Matplotlib 把結果畫成圖表，比較容易看出來。我畫了兩個子圖，一個是函數比較圖，一個是誤差分析圖，都用對數尺度來顯示，這樣可以同時看到小誤差和大誤差的變化。

3.1 實驗設置

這次實驗我設定了以下參數：

- x 的範圍：從 -2π 到 2π，共 1000 個點
- 測試的項數：$n = 1, 3, 5, 8$
- 誤差閾值：0.01（用於進階分析）

3.2 為什麼使用向量化運算？

向量化運算是 NumPy 的重點功能，好處是：

1. **算得比較快：**
    - Python 的迴圈很慢，向量化運算可以一次處理很多資料
    - NumPy 底層是用 C 寫的，所以很快
    - 可以一次對很多數字做同樣的運算

2. **程式碼比較簡單：**不用寫迴圈，程式碼看起來比較清楚。

3. **比較省記憶體：**不需要建立很多中間變數。

4 數據分析與視覺化 (Data Analysis & Visualization)

4.1 逼近曲線分析

從圖表（圖 1 左側）可以看出：

- **n=1（線性逼近）：**這是 $y = x$ 的直線，只有在 x 很小的時候才準，離 0 遠一點就開始偏了。當 $|x| > 0.5$ 時，誤差就明顯可見。
- **n=3（三次逼近）：**加入了 x^3 項，曲線開始有彎曲的形狀，範圍變大一點，大概到 $|x| < \pi$ 都還算準。但超過這個範圍後，誤差會快速增加。
- **n=5（五次逼近）：**逼近效果更好，在 $|x| < 2\pi$ 的範圍內都能很好地逼近 $\sin(x)$。
- **n=8（八次逼近）：**範圍更大，跟真正的 $\sin(x)$ 幾乎重疊，在整個 $[-2\pi, 2\pi]$ 範圍內都很準確。
- **結論：**項數越多，能準確逼近的範圍就越大。從圖中可以清楚看到，隨著項數增加，逼近曲線越來越接近真實的 $\sin(x)$ 曲線。

4.2 誤差行為分析

為了看清楚誤差有多大，我畫了**絕對誤差圖**，用對數尺度來看（圖 1 右側）：

- **靠近 0 的地方：**誤差很小，大概 10^{-15} 左右，幾乎是電腦能算到最準的程度了。這表示在 $x = 0$ 附近，泰勒級數的逼近效果非常好。
- **離 0 越遠：**誤差就越大，而且長得很快。從圖中可以看到誤差呈現指數級成長的趨勢。
- **項數的取捨：**項數多一點誤差會比較小，但算的時間也會變長，這就是準確度和速度之間的取捨。
- **誤差閾值分析：**根據程式的進階分析，當誤差閾值設為 0.01 時，n=1 項只能在 $|x| < 0.38$ 的範圍內達到要求；n=3 項可以擴展到 $|x| < 1.75$；n=5 項可以到 $|x| < 3.24$；n=8 項則可以達到 $|x| < 5.49$。這清楚地展示了項數增加對誤差控制的影響。

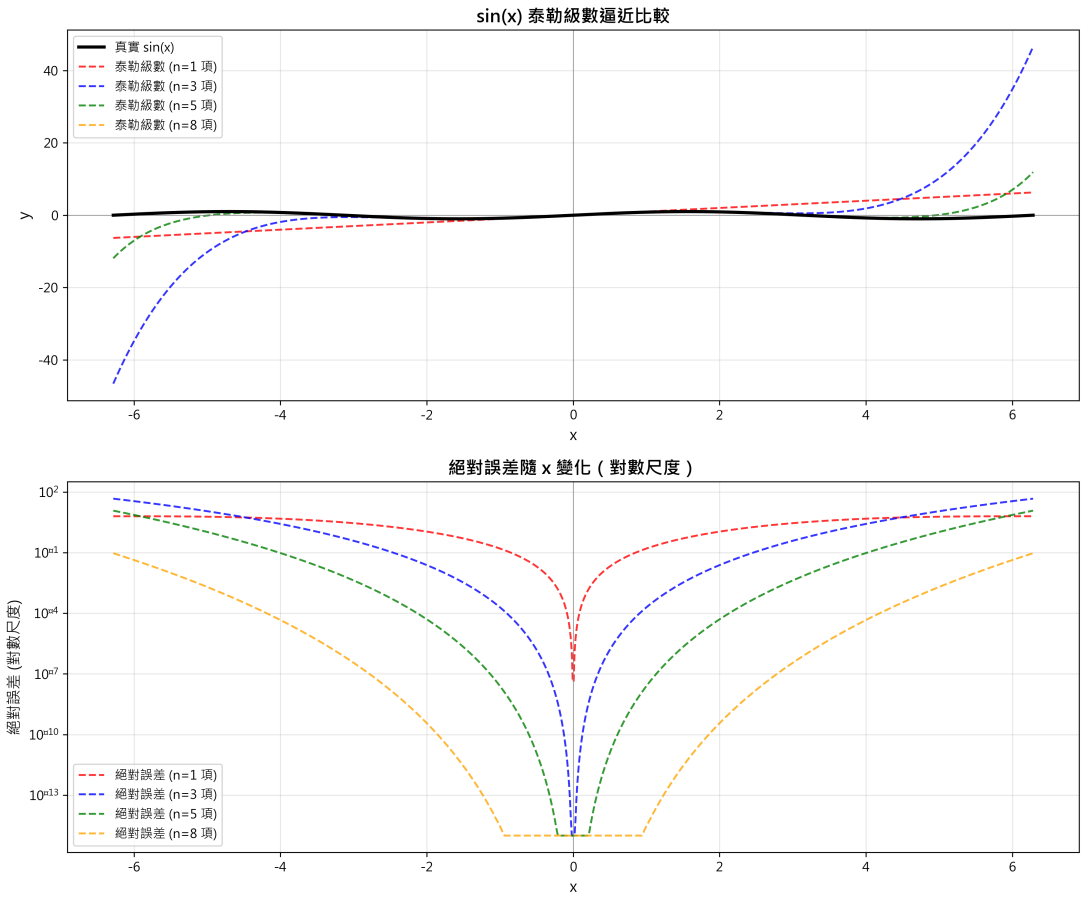

圖 1: sin(x) 泰勒級數逼近比較與誤差分析圖

5 結論 (Conclusion)

這次專案讓我用 Python 把微積分的理論實際跑了一遍，從中學到了很多。

1. **理論驗證：**結果跟學的泰勒級數一樣，證明用多項式來逼近是可行的。實驗結果清楚地展示了泰勒級數的收斂特性，項數越多逼近效果越好。

2. **工具學習：**學會用 NumPy 和 SciPy，發現它們比直接用 Python 的列表快很多。向量化運算讓程式碼更簡潔，執行效率也大幅提升。

3. **實用經驗：**知道在寫程式時，要根據需要的準確度和計算時間來決定要用幾項，這是這次學到的重要經驗。不同的應用場景需要不同的精度，不能一味追求高精度而忽略計算成本。

4. **視覺化的重要性：**透過圖表可以很直觀地看到誤差的變化趨勢，這比單純看數字更容易理解。對數尺度的使用讓小誤差和大誤差都能清楚呈現。

5.1 精確度與計算成本的平衡

從實驗結果可以得出幾個結論：

1. **項數和準確度的關係：**
    - 項數越多算得越準，能用的範圍也越大
    - 但是項數多就要算比較久
2. x **的範圍影響：**
    - 如果 x 很小（靠近 0），用幾項就很準了
    - 如果 x 比較大，就要用更多項才能維持準確度
3. **實際應用建議：**
    - 如果 $|x| < 1$：用 3-5 項就夠了，誤差可以控制在很小的範圍內。
    - 如果 $|x| < 2\pi$：建議用 8-10 項，這樣可以保證在整個範圍內都有良好的精度。
    - 如果範圍更大：需要更多項，或者考慮其他方法，比如先將 x 縮小到 $[-\pi, \pi]$ 範圍內再計算。
4. **未來改進方向：**
    - 可以嘗試使用其他數值方法，如 CORDIC 演算法，在某些情況下可能更有效率。
    - 可以實作自適應項數選擇，根據輸入的 x 值自動決定需要多少項。
    - 可以比較不同逼近方法的計算效率和精度。