第9章 Python 進階

9.1 Json

JSON（JavaScript Object Notation）是一種輕量級的數據交換格式。它基於JavaScript的一個子集，但是許多程式語言都有JSON格式數據的讀寫能力。JSON格式被廣泛用於網絡應用之間的數據交換，以及配置文件和數據儲存等場景。

JSON的基本結構

JSON主要有兩種結構：

1. **對象（Object）**：由花括號 `{}` 包圍，表示一組由鍵（Key）和值（Value）組成的無序集合。鍵是字符串，值可以是字符串、數字、布爾值、數組、對象或null。

例如：

```
{
  "name": "John",
  "age": 30,
  "married": true
}
```

2. **數組（Array）**：由方括號 `[]` 包圍，表示一組有序的值列表，值的類型可以是字符串、數字、布爾值、數組、對象或null。

例如：

```
[ "apple", "banana", "cherry" ]
```

JSON與Python

在Python中，可以使用內建的 `json` 模組來處理JSON數據。`json` 模組提供了以下主要功能：

- `json.loads()`：將JSON字符串解析成Python對象。
- `json.dumps()`：將Python對象轉換成JSON格式的字符串。

例如，將Python字典轉換為JSON字符串：

```
import json

data = {
  "name": "John",
  "age": 30,
  "married": True
}

json_string = json.dumps(data)
print(json_string)
```

這將輸出一個JSON格式的字符串。

JSON因其簡潔和易於理解的格式，成為網絡交換數據的流行選擇。

9.2 正規表示式

正規表示式（Regular Expressions，簡稱 Regex）是一種強大的工具，用於匹配和處理文本。它們通過一系列符號和字符來描述或匹配一系列符合某個語法規則的字符串。正規表示式廣泛應用於字符串搜索、替換、數據驗證等場景。

基本概念

1. **文字字符** ：普通的字符表示它們自己，例如 'a' 匹配 'a'。
2. **元字符** ：具有特殊含義的字符，如 `.`（匹配任意單個字符）、`^`（匹配行開始）、`$`（匹配行結尾）等。
3. **字符集** ：用 `[]` 表示，匹配括號內的任何字符。例如，`[abc]` 匹配 'a'、'b' 或 'c'。
4. **量詞** ：指定字符出現的次數。如 `*`（零次或多次）、`+`（一次或多次）、`?`（零次或一次）。
5. **分組和引用** ：用 `()` 進行分組，可以記錄一部分正則表達式匹配的文本。
6. **或操作** ：用 `|` 表示，匹配左邊或右邊的表達式。
7. **轉義字符** ：用 `\` 將特殊字符轉化為普通字符，或相反，例如 `\d` 匹配任何數字。

常見用途

1. **驗證輸入** ：檢查字符串是否符合特定格式（如電子郵件、電話號碼）。
2. **搜索和替換** ：在文本中查找或替換符合特定模式的字符串。
3. **數據提取** ：從文本中提取信息，如從日誌文件中提取特定數據。
4. **文本分析** ：用於語言處理和文本數據挖掘。

Regex101

Regex101 是一個在線工具，它提供了一個強大的界面來創建、測試和調試正則表達式（Regular Expressions，簡稱 regex）。這個工具被廣泛用於開發和學習正則表達式，特別是因為它提供了豐富的功能和直觀的用戶界面。以下是 Regex101 的一些主要特點：

1. **多語言支持** ：Regex101 支持多種編程語言的正則表達式語法，包括 Python、JavaScript、PHP 和 Go。
2. **實時匹配信息** ：當你輸入正則表達式和測試字符串時，Regex101 會實時顯示匹配結果，這有助於理解正則表達式的行為。
3. **詳細的解釋** ：這個工具對正則表達式的每一部分提供詳細的解釋，這有助於初學者學習和理解各種構造和模式。
4. **調試器** ：Regex101 包含一個正則表達式調試器，可以顯示正則表達式在匹配過程中的每一步。
5. **單元測試** ：用戶可以為他們的正則表達式創建和運行單元測試，這有助於確保正則表達式的正確性和穩健性。
6. **社區庫** ：Regex101 有一個共享庫，用戶可以在這裡發布他們的正則表達式，並瀏覽其他用戶創建的表達式。
7. **代碼生成** ：它還可以根據用戶的正則表達式生成相應語言的代碼片段。

Regex101 是一個非常有用的資源，對於想要學習正則表達式或需要在他們的項目中使用正則表達式的開發人員來說，是一個寶貴的工具。

9.3 鏈式調用

當我們談到程式設計中的「鏈式調用」（Chaining），其實就是一種讓代碼更簡潔、連貫的寫法。想像一下，你有一串珍珠，每顆珍珠都連接著下一顆。在程式設計中，我們也可以用類似的方式來「連接」一系列的操作。

讓我們用一個簡單的比喻來理解鏈式調用：想像你在一家餐廳，你的點餐過程可以是這樣的：

1. **點餐** ：你先告訴服務員你要一份主餐。
2. **加配料** ：接著你加點一些配料。
3. **選飲料** ：最後，你再選一杯飲料。

在傳統的程式設計中，這可能需要分成幾個步驟來寫：

```
order = Restaurant()
order.select_main_course("漢堡")
order.add_side_dish("薯條")
order.choose_drink("可樂")
```

但是，如果使用鏈式調用，你可以將這些步驟「鏈接」起來，變成一行流暢的代碼：

```
order = Restaurant().select_main_course("漢堡").add_side_dish("薯條").choose_dr
```

這裡的 `select_main_course` 、 `add_side_dish` 、 `choose_drink` 都是餐廳物件的方法。每個方法都執行一個動作，然後返回餐廳物件本身，這樣下一個方法就可以接著調用。

使用鏈式調用的好處是讓代碼看起來更簡潔、易讀。它將一系列相關的操作連接在一起，避免了反覆提及同一個物件。這種方式在許多程式語言中都很常見，尤其是在需要進行一系列操作的場景中。在 Python 中，這種寫法同樣適用，特別是在處理字符串、數據結構操作或者建立複雜的查詢時。

9.4 類別之魔術方法

在 Python 中，使用類（class）和魔術方法（magic methods，也稱為特殊方法）可以為自定義的資料結構提供直觀且強大的操作方式。以向量運算為例，我們可以定義一個表示三維向量的類，並通過實現魔術方法來使得向量運算更加直觀和方便。

類（Class）的好處：

1. **封裝性** ：將數據（如 x, y, z 坐標）和與之相關的方法（如向量加法、點積）封裝在一起，使得代碼結構更加清晰和模組化。
2. **重用性** ：一旦定義了向量類，就可以在不同的地方多次創建和使用向量實例，無需重複代碼。
3. **易於維護和擴展** ：如果需要添加新的功能或修改現有功能，只需更改類的定義，不影響使用該類的其他代碼。

魔術方法的好處：

魔術方法是 Python 中特殊的方法，由雙下劃線（如 `__init__` 、 `__add__` ）包圍的方法名稱標識。在向量類中實現這些方法有以下好處：

1. **直觀的運算子重載** ：通過實現如 `__add__` （加法）、 `__sub__` （減法）、 `__mul__` （乘法）等方法，可以使得向量對象支持 `+` 、 `-` 、 `*` 等運算子，使代碼更加直觀。
2. **自定義的打印格式** ：通過實現 `__str__` 或 `__repr__` 方法，可以定義向量對象的打印格式，使其更有閱讀性。
3. **更多的操作支持** ：實現如 `__len__` （返回向量長度）或 `__getitem__` （支持索引操作）等方法，可以讓向量類的實例擁有更多類似內建數據類型的特性和功能。

實例：三維向量類

下面是一個簡單的示例，展示如何使用類和魔術方法來定義一個三維向量類：

```python
class Vector3D:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def __add__(self, other):
        return Vector3D(self.x + other.x, self.y + other.y, self.z + other.z)

    def __sub__(self, other):
        return Vector3D(self.x - other.x, self.y - other.y, self.z - other.z)

    def __mul__(self, scalar):
        return Vector3D(self.x * scalar, self.y * scalar, self.z * scalar)

    def __matmul__(self, other):
        return self.x * other.x + self.y * other.y + self.z * other.z

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y and self.z == other.z

    def __abs__(self):
        return (self.x**2 + self.y**2 + self.z**2)**0.5

    def __repr__(self):
        return f"Vector3D({self.x}, {self.y}, {self.z})"

    def __str__(self):
        return f"({self.x}, {self.y}, {self.z})"
```

以下是這些方法的簡要說明和實際應用：

1. **初始化 (`__init__`)**：設置向量的 x, y, z 坐標。
2. **加法 (`__add__`)**：實現兩個向量的加法，返回它們的和。
3. **字符串表示 (`__str__`)**：提供向量的友好字符串表示，適用於打印和調試。
4. **減法 (`__sub__`)**：實現兩個向量的減法，返回它們的差。

5. **乘法 (`__mul__`)**：允許向量與標量的乘法，返回標量乘以向量後的結果。
6. **向量點積 (`__matmul__`)**：通過 `@` 運算子實現兩個向量的點積，返回一個標量值。
7. **等於 (`__eq__`)**：判斷兩個向量在各坐標上是否相等。
8. **絕對值 (`__abs__`)**：計算並返回向量的模（長度）。
9. **正式字符串表示 (`__repr__`)**：提供向量的正式字符串表示，通常用於開發和調試階段。
   #####使用示例

```
# 創建兩個向量實例
v1 = Vector3D(1, 2, 3)
v2 = Vector3D(4, 5, 6)

# 向量加法
print("加法:", v1 + v2)  # 輸出: 加法: (5, 7, 9)

# 向量減法
print("減法:", v1 - v2)  # 輸出: 減法: (-3, -3, -3)

# 向量乘以標量
print("純量乘法:", v1 * 3)  # 輸出: 純量乘法: (3, 6, 9)

# 向量點積
print("點積:", v1 @ v2)  # 輸出: 點積: 32

# 比較向量是否相等
print("是否相等:", v1 == Vector3D(1, 2, 3))  # 輸出: 是否相等: True

# 向量的模
print("向量v1的模:", abs(v1))  # 輸出: 向量v1的模: 3.7416573867739413
```

這些魔術方法的實現不僅提升了 `Vector3D` 類的可用性和靈活性，還增加了其與 Python 內建數據類型的一致性，使得使用者能夠以更自然和直觀的方式使用向量。

9.5 Matplotlib

Matplotlib 是一個廣泛使用的 Python 繪圖庫，非常適合用於數據可視化。它提供了一個功能豐富的繪圖界面，用於繪製靜態、動態以及互動式的圖表。Matplotlib 可以與 NumPy 和 SciPy 等數學計算庫結合使用，方便在科學計算中展示結果。

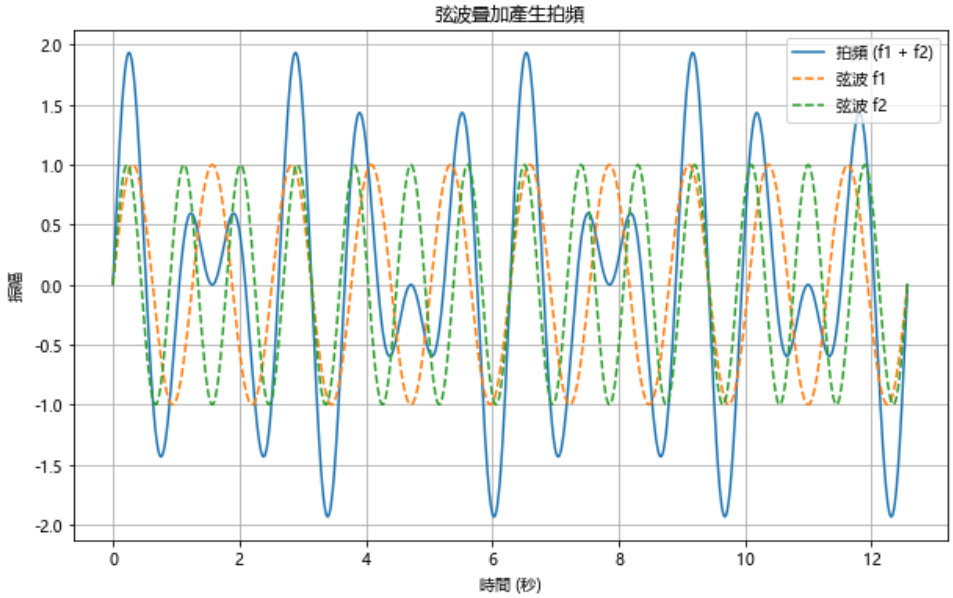
Matplotlib 的主要特點包括：

1. **多樣的繪圖類型** ：支持線圖、散點圖、柱狀圖、餅圖、直方圖、箱形圖等多種圖表類型。
2. **高度可定制** ：幾乎每個圖表元素（如顏色、大小、標籤）都可定制，以符合具體的展示需求。
3. **支持多種格式** ：可以將圖表保存為多種格式，包括 PNG、PDF、SVG 等。
4. **交互式環境** ：在 Jupyter Notebook 等交互式環境中使用時，可以動態更新和交互圖表。

Matplotlib 常用於數據分析、科學研究、工程設計等領域，特別是在需要將數據視覺化呈現給用戶時。

範例：拍頻現象

以下這張圖展示了兩個不同頻率的弦波（分別為5 Hz和7 Hz）疊加後產生的拍頻效果。您可以看到，當這兩個弦波合併時，它們的振幅會互相影響，導致圖中的波形出現周期性的加強和減弱。這就是拍頻現象，它是由於兩個頻率接近但不完全相同的波形相互作用造成的。



不同頻率弦波疊加產生拍頻

在圖中，實線表示兩個波形相加的結果（即拍頻），而虛線分別代表了原始的兩個弦波。可以觀察到，當這兩個波形在某些點相位一致時，它們相加造成振幅增加；當它們相位相反時，則相互抵消，導致振幅減小。這種現象在聲學和物理學中非常常見，並且對於理解波的干涉和疊加非常有幫助。

```python
import numpy as np

import matplotlib.pyplot as plt


# 設置支持中文的字體
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']  # 微软雅黑作为例子
plt.rcParams['axes.unicode_minus'] = False  # 正確顯示負號

# 設定時間範圍
t = np.linspace(0, 4*np.pi, 1000)

# 創建兩個頻率略有不同的弦波
f1 = 5  # 頻率 5 Hz
f2 = 7  # 頻率 5.2 Hz
y1 = np.sin(f1 * t)
y2 = np.sin(f2 * t)

# 疊加這兩個弦波產生拍頻
y = y1 + y2

# 繪製圖形
plt.figure(figsize=(10, 6))
plt.plot(t, y, label='拍頻 (f1 + f2)')
plt.plot(t, y1, '--', label='弦波 f1')
plt.plot(t, y2, '--', label='弦波 f2')
plt.title('弦波疊加產生拍頻')
plt.xlabel('時間 (秒)')
plt.ylabel('振幅')
plt.legend()
plt.grid(True)
plt.show()
```

9.6 Numpy

Numpy 是一個在 Python 中廣泛使用的數學計算庫，尤其在數據分析、科學計算和工程領域中非常受歡迎。它提供了強大的多維數組對象（array）和相關的大量數學運算函數。使用 Numpy，您可以進行矩陣運算、統計分析、數據轉換等複雜任務。

一些常見的 Numpy 功能包括：

1. 數組操作：創建、修改、索引、切片和操作多維數組。
2. 數學計算：支持基本數學運算、線性代數、傅立葉變換等。
3. 統計函數：計算平均值、中位數、標準差等統計指標。
4. 隨機數生成：產生各種概率分佈下的隨機數。

實際例子

在物理學中，電場線是用來表示電場方向和強度的虛擬線條。當有兩個帶電荷的物體，比如一個帶正電荷（+Q）和一個帶負電荷（-Q），它們會在周圍空間產生電場，這些電場線可以從正電荷指向負電荷。

要用 Python 和 NumPy 庫來計算和繪製這樣的電場線，可以遵循以下步驟：

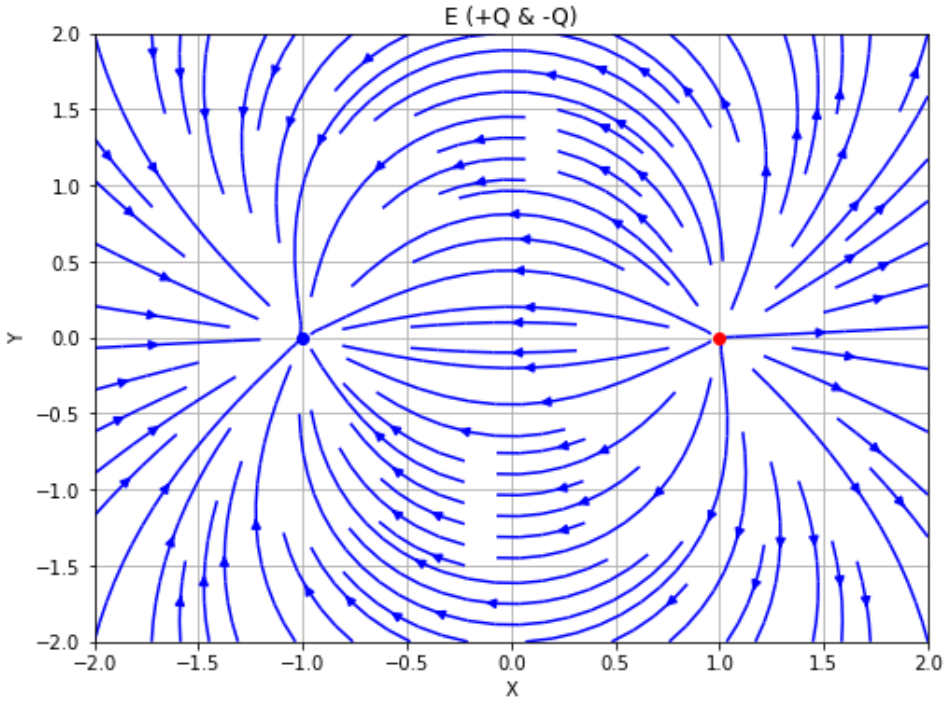
1. 定義電荷的位置和大小。
2. 在感興趣的區域建立一個網格。
3. 對於網格中的每一點，計算由兩個電荷產生的電場向量。
4. 繪製電場線。

現在我會用 Python 代碼來示範這個過程。我將假設 +Q 和 -Q 位於二維空間中，並且它們的電荷量相等但符號相反。

```python
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

# 定義電荷的參數
Q = 1e-9  # 電荷量 (庫倫)
k = 8.988e9  # 靜電力常數 (N·m²/C²)

# 定義電荷的位置
pos_charge = np.array([1.0, 0.0])  # 正電荷的位置
neg_charge = np.array([-1.0, 0.0])  # 負電荷的位置

# 建立一個網格
x = np.linspace(-2, 2, 100)
y = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(x, y)

# 計算每個點的電場
def electric_field(Q, r0, x, y):
    """ 計算單個點的電場 """
    r = np.sqrt((x - r0[0])**2 + (y - r0[1])**2)
    Ex = k * Q * (x - r0[0]) / r**3
    Ey = k * Q * (y - r0[1]) / r**3
    return Ex, Ey

Ex, Ey = electric_field(Q, pos_charge, X, Y)
Ex2, Ey2 = electric_field(-Q, neg_charge, X, Y)

# 疊加電場
total_Ex = Ex + Ex2
total_Ey = Ey + Ey2

# 繪製電場線
plt.figure(figsize=(8, 6))
plt.streamplot(X, Y, total_Ex, total_Ey, color='b')
plt.plot(pos_charge[0], pos_charge[1], 'ro')  # 繪製正電荷
plt.plot(neg_charge[0], neg_charge[1], 'bo')  # 繪製負電荷
plt.title('E (+Q & -Q)')
```

```
plt.xlabel('X')
plt.ylabel('Y')
plt.grid()
plt.show()
```

執行計算結果



電力線分布計算

這幅圖展示了由一個帶正電荷（+Q）和一個帶負電荷（-Q）產生的電場線。在這裡，正電荷用紅色點表示，負電荷用藍色點表示。電場線從正電荷出發，指向負電荷，顯示了電場的方向和相對強度。這個圖示是使用 Python 和 NumPy 來計算的，顯示了在不同位置的電場向量如何受到這兩個電荷的影響。

9.7 Scipy及優化

SciPy 是一個基於 Python 的開源軟件，用於數學、科學和工程計算。它建立在 NumPy 的基礎上，提供了大量的模塊和函數，用於專門的數學和科學計算。與 NumPy 的基本數組操作不同，SciPy 包含更多針對特定科學領域的高級算法和功能。

SciPy 主要特點包括：

1. **優化和求解** ：提供了函數最小化、曲線擬合和求解方程的工具。
2. **積分** ：提供了數值積分和常微分方程求解器。
3. **線性代數** ：包括矩陣分解和其他高階線性代數運算。
4. **統計** ：提供了概率分佈、統計測試和描述統計的功能。
5. **信號處理** ：用於信號濾波、平滑和變換等。
6. **圖像處理** ：提供了基本的圖像處理工具，如濾波、形態學操作等。

優化

優化是數學和工程領域中一個重要的概念，其目的是在給定的條件下找到某個函數的最大值或最小值。這個函數通常被稱為「目標函數」或「損失函數」，而條件則被稱為「約束條件」。優化問題無處不在，從日常生活的決策問題到工程設計、經濟學、物理學等專業領域都會遇到。

優化的類型

1. **線性優化** ：目標函數和約束條件都是線性的。
2. **非線性優化** ：目標函數或約束條件至少有一個是非線性的。
3. **整數優化** ：解必須是整數。
4. **組合優化** ：尋找最佳的組合解。

優化的應用

- **工程設計** ：如機械零件的最優尺寸設計。
- **經濟學** ：資源分配、風險管理等。
- **人工智慧** ：機器學習中的模型訓練。
- **物流和運籌學** ：如最短路徑問題、庫存管理。

優化的方法

- **解析方法** ：透過數學公式直接求解。
- **數值方法** ：透過迭代算法逼近解，例如梯度下降法、單純形法。
- **啟發式方法** ：用於解決複雜的優化問題，例如遺傳算法、模擬退火法。

優化的挑戰

- **局部最優與全局最優** ：尤其在非線性優化中，找到的解可能是局部最優而非全局最優。
- **計算複雜度** ：某些優化問題計算成本非常高。
- **約束條件的處理** ：如何有效地處理和整合約束條件。

總之，優化是尋找最佳解決方案的科學，無論是在理論還是實踐層面，它都具有極高的重要性和廣泛的應用範圍。

`scipy.minimize` 是 SciPy 庫中的一個功能強大的優化函數，用於尋找多變數函數的局部最小值。它非常適合於解決複雜的非線性優化問題，包括有約束和無約束的情況。在您提供的例子中，目標是最小化一個四元非線性方程式，同時考慮約束條件和變數的範圍限制。

範例

目標方程式是一個非線性函數，形式如下：

$$f(x) = x1 \cdot x4 \cdot (x1 + x2 + x3) + x3$$

其中 $x1, x2, x3, x4$ 是變數。

約束條件

您的問題包含兩個約束條件：

1. **非線性不等式約束** ：$x1 \cdot x2 \cdot x3 \cdot x4 \geq 25$
2. **非線性等式約束** ：$x_1^2 + x_2^2 + x_3^2 + x_4^2 = 40$

變數範圍

每個變數的範圍都被限制在 1 到 5 之間。

使用 scipy.minimize

要使用 `scipy.minimize` 解決這個問題，您需要定義目標函數和約束條件，並指定變數的範圍。以下是一個示例代碼：

```
import numpy as np
from scipy.optimize import minimize

# 目標函數
def objective(x):
    return x[0] * x[3] * (x[0] + x[1] + x[2]) + x[2]

# 約束條件
def constraint1(x):
    return x[0] * x[1] * x[2] * x[3] - 25

def constraint2(x):
    return 40 - (x[0]**2 + x[1]**2 + x[2]**2 + x[3]**2)

# 變數範圍
bnds = ((1, 5), (1, 5), (1, 5), (1, 5))

# 約束類型
con1 = {'type': 'ineq', 'fun': constraint1}
con2 = {'type': 'eq', 'fun': constraint2}

# 初始猜測值
x0 = [1, 5, 5, 1]

# 執行優化
sol = minimize(objective, x0, method='SLSQP', bounds=bnds, constraints=[con1, c

print(sol)
```

在這段代碼中，我們使用了序列二次規劃（Sequential Least SQuares Programming，SLSQP）方法，這是解決帶有約束條件的非線性優化問題的一種常見方法。

輸出結果

```
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 17.01401724556073
       x: [ 1.000e+00  4.743e+00  3.821e+00  1.379e+00]
     nit: 5
     jac: [ 1.457e+01  1.379e+00  2.379e+00  9.564e+00]
    nfev: 25
    njev: 5
```

以下是對這些輸出參數的解釋：

1. **message** : "Optimization terminated successfully" - 這表明優化過程成功結束。
2. **success** : True - 這表明優化過程成功找到了一個解。
3. **status** : 0 - 在 SciPy 中，狀態碼 0 通常表示成功。
4. **fun** : 17.01401724556073 - 這是目標函數在最優解處的值。對於最小化問題，這是找到的最小值。
5. **x** : [1.000e+00, 4.743e+00, 3.821e+00, 1.379e+00] - 這是找到的最優解。對於您的問題，這表示 x1, x2, x3, 和 x4 的最佳值。
6. **nit** : 5 - 這表示優化器進行了 5 次迭代才找到最優解。
7. **jac** : [14.57, 1.379, 2.379, 9.564] - 這是目標函數的雅可比矩陣在最優解處的估計。雅可比矩陣表示目標函數對每個變數的一階偏導數。
8. **nfev** : 25 - 這表示目標函數被評估了 25 次。
9. **njev** : 5 - 這表示雅可比矩陣被評估了 5 次。

這些資訊綜合提供了關於優化過程的效率和結果的詳細洞察。成功的 `success` 標記和 "Optimization terminated successfully" 消息表明優化問題已經成功解決。

9.8 scikit-rf模組

`scikit-rf`（通常縮寫為 `skrf`）是一個用於射頻（RF）和微波工程的開源Python庫。這個庫提供了豐富的工具和功能，專為RF和微波工程的需求設計。這些功能包括但不限於對S參數（散射參數）的處理、網絡分析、數據可視化，以及與常見RF測量設備的交互。

主要功能

1. **S參數處理** ：`scikit-rf` 支持對S參數的讀取、寫入、計算和可視化，這對於分析和設計RF電路特別重要。
2. **網絡分析** ：提供了用於操作和分析網絡參數（如S參數、Z參數、Y參數等）的功能，包括網絡合成、轉換和其他操作。
3. **數據可視化** ：內置多種數據可視化工具，可以方便地生成參數圖表，例如史密斯圖（Smith Chart）和尼柯斯圖（Nyquist Plot）。
4. **測量校準** ：提供了一套完整的測量校準工具，用於校準RF測量數據。
5. **與儀器交互** ：支持與常見的RF測量設備交互，例如網絡分析儀。

安裝

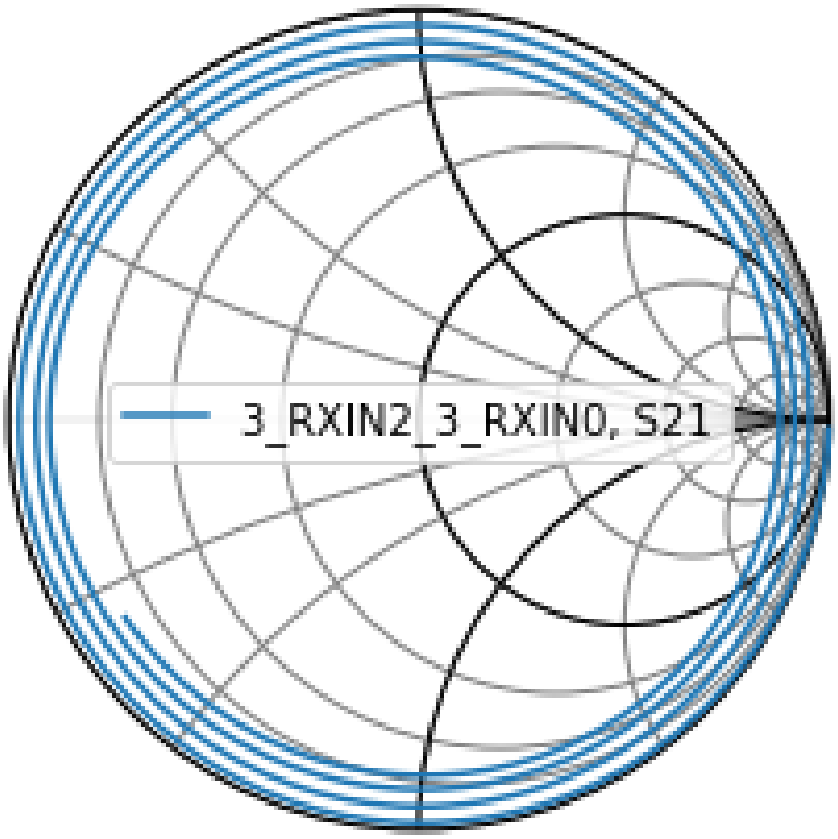
`scikit-rf` 可以通過Python的包管理器pip安裝：

```
pip install scikit-rf
```

使用示例

下面是一個簡單的示例，展示了如何使用 `scikit-rf` 讀取S參數文件並繪製史密斯圖：

```
import skrf as rf
import matplotlib.pyplot as plt

# 加載S參數數據
ntwk = rf.Network('your_s2p_file.s2p')

# 繪製S11參數的史密斯圖
ntwk.plot_s_smith(m=0, n=0)
plt.show()
```



Smith Chart

在這個例子中，`your_s2p_file.s2p` 應該替換為包含您的S參數數據的文件路徑。這段代碼將讀取文件並繪製S11參數的史密斯圖。

總的來說，`scikit-rf` 是射頻和微波工程師在進行相關計算和分析時的有力工具。