第4章 資料處理範例

在使用AEDT進行模擬時，幾乎所有的模擬結果都可以以CSV（逗號分隔值）格式導出。因此，掌握如何有效地處理CSV文件是進行模擬數據分析的基礎技能。儘管CSV是一種廣泛使用的數據格式，但理解如何從CSV文件中讀取數據並將其有效地轉換為變數，是對數據進行進一步處理和分析的關鍵步驟。

使用 Python 處理大量數據的模擬檔案有許多優勢。Python 的這些優點包括：

1. **靈活性和擴展性** ：Python 支持多種數據結構和格式，可以輕鬆地處理和分析各種類型的數據。此外，Python 可以與其他語言和工具（如 C/C++、Java、R）集成，進一步擴展其功能。
2. **豐富的庫和框架** ：Python 擁有豐富的庫和框架，如 NumPy、Pandas、Matplotlib、SciPy、Scikit-learn 等，這些工具對於數據分析和可視化非常有用。
3. **數據處理和分析的高效性** ：Python 在數據處理和分析方面表現出色，特別是當處理大規模數據集時，它的性能表現仍然非常好。
4. **自動化和重複使用** ：Python 腳本可以輕鬆地自動化繁瑣的數據處理任務，並且可以重複使用於不同的數據集和專案中。

總之，Python 在數據處理和分析領域提供了一個強大、靈活且用戶友好的環境，非常適合處理大量的模擬數據檔案。

4.1 CSV檔案處理

CSV（逗號分隔值）檔案格式是一種常用的文本格式，用來儲存表格數據。它的特點是簡單易用，可以被多種軟件讀取，例如Microsoft Excel、各種文字編輯器，甚至是編程語言中的數據庫操作函數。

CSV檔案格式

一個CSV檔案通常由以下元素組成：

1. **數據行** ：每行代表表格中的一條記錄。
2. **逗號** ：作為分隔符號，用來分隔一行中的各個字段。
3. **字段** ：每條記錄的個別數據單元，可以是文字、數字等。
4. **換行符** ：用於分隔各個記錄。

例如，一個簡單的CSV檔案可能看起來像這樣：

```
姓名,年齡,職業
張三,30,工程師
李四,25,設計師
王五,28,教師
```

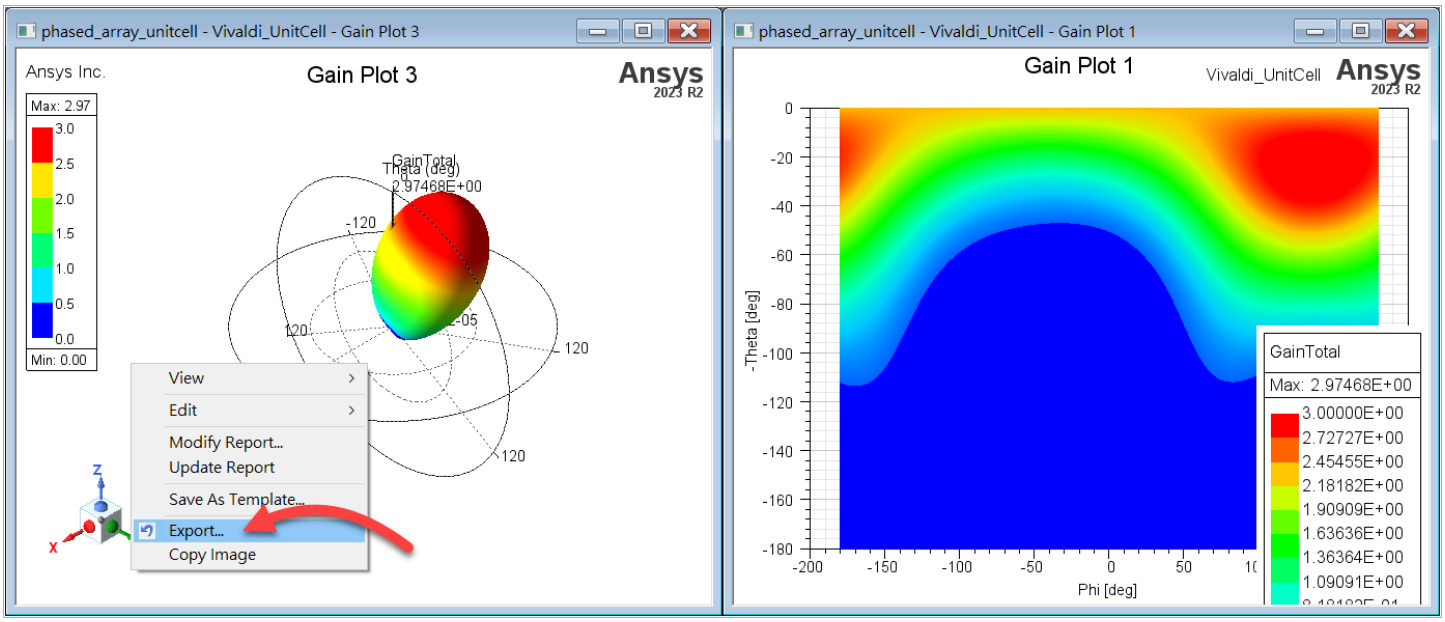
這個例子中，每行代表一個人的資料，包括姓名、年齡和職業，而這些資料則由逗號分隔。CSV格式的簡潔性和通用性使它成為數據交換的常見選擇。

挑戰：找到最大Gain值及對應之角度

這裡展示的是 HFSS 模擬的 3D 遠場結果。左邊的圖呈現了 3D 遠場的全面視角，可以讓我們移動視角從不同方向觀察場的分佈。而右邊的圖則採用等高線圖（Contour）的方式呈現，這種表示法能夠清晰地展示不同增益值在 phi 和 theta 兩個角度上的分佈。在這張等高線圖中，橫軸代表 phi 角度，縱軸代表 theta 角度。透過這樣的視覺呈現，我們能更直觀地理解不同方向上的遠場特性。

若想右邊圖表上找到並標記最大值，過程可能相當手動且耗時。首先，我們需要啟用標記功能（mark）。然後，基於對數據的初步理解，比如預期最大值可能出現在右上角，我們手動移動滑鼠到這一區域。隨著滑鼠的移動，我們需密切觀察數值的變化。

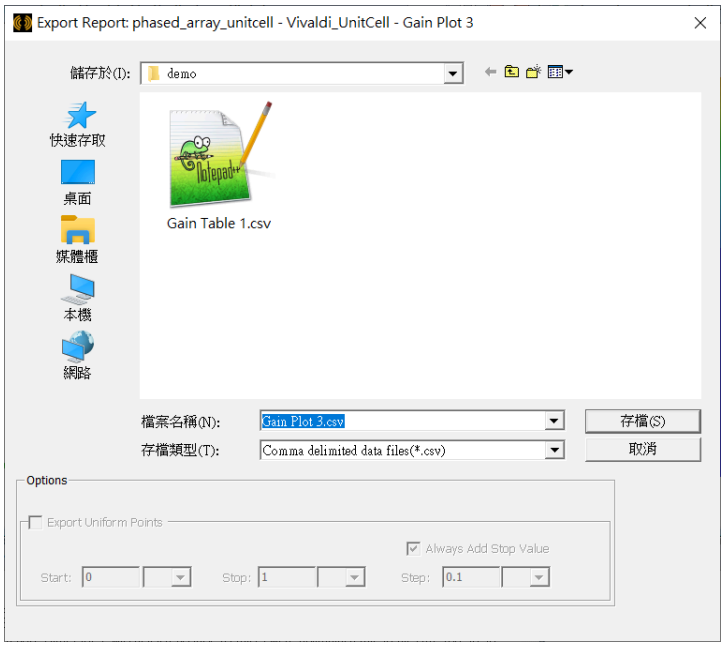
當找到顯示最大值的點時，我們在該位置放置標記。這一過程需要持續調整和觀察，直到確信找到了真正的最大值。這種方法在數據複雜或圖表範圍廣泛時尤其困難，容易造成時間的浪費和精度上的不確定性。



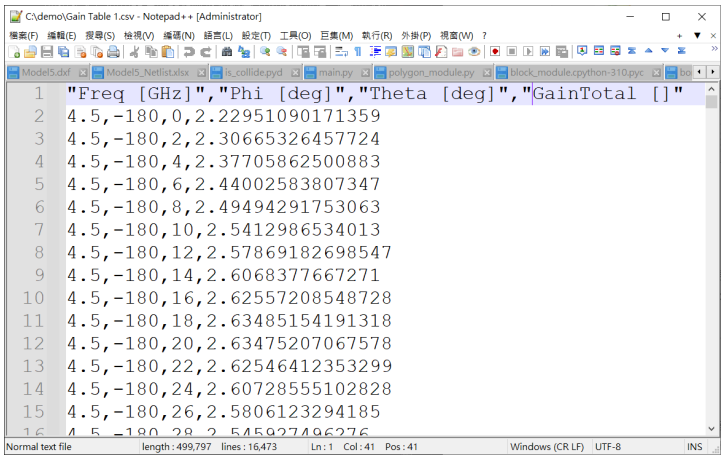
左圖:3D遠場/右圖:熱圖表示法

匯出CSV檔案

要將圖表中的數據匯出為 .csv 檔案，非常簡單方便。只需在圖表上任意位置點擊右鍵，就會出現一個選單。選擇Export之後，您可以將圖表中的所有數據保存為 .csv 格式的檔案。這個功能讓您能夠輕鬆地將圖表數據用於進一步的分析或報告製作。



儲存csv檔案



在文字編輯器當中檢視csv檔案內容

這個檔案主要包含四個欄位，分別為：

1. **"Freq [GHz]"** ：這個欄位代表頻率，單位是吉赫茲（GHz）。
2. **"Phi [deg]"** ：表示 Phi 角度，單位是度（deg）。Phi 通常指的是在極坐標系統中的方位角。
3. **"Theta [deg]"** ：表示 Theta 角度，單位也是度。Theta 在極坐標系統中通常指的是天頂角。
4. **"GainTotal []"** ：這個欄位代表總增益，似乎沒有特定的單位。

CSV 檔案中的每一行都代表一個特定的數據點，其中包含了在特定頻率、Phi 角度和 Theta 角度下的總增益值。例如，第一行數據 "4.5,-180,0,2.22951090171359" 表示在 4.5 GHz 頻率、-180度 Phi 角度和 0度 Theta 角度下的總增益為 2.22951090171359。

####完整程式碼
以下程式碼是用來從 CSV 檔案中讀取數據，並將這些數據視覺化為一個熱力圖，同時標記出增益最大值的位置。這裡是代碼的步驟和功能解釋：

1. **讀取數據** ：

- 使用 `open` 函數打開指定路徑的 CSV 檔案。
- 使用 `readlines` 方法將檔案內容讀取為一個列表，每一行為列表的一個元素。

2. **提取和排序數據** ：

- 透過迴圈處理每一行數據，分別提取 phi、theta 和 gain_total 值。
- 使用集合 `set` 來儲存唯一的 phi 和 theta 值，然後轉換為有序列表。

3. **創建二維數組** ：

- 將 gain_total 列表轉換成 numpy 二維數組 `z_2d`。此數組的形狀由 phi 和 theta 的數量決定。

4. **尋找最大值和位置** ：

- 使用 numpy 函數找出 `z_2d` 中的最大值，以及該值的位置。

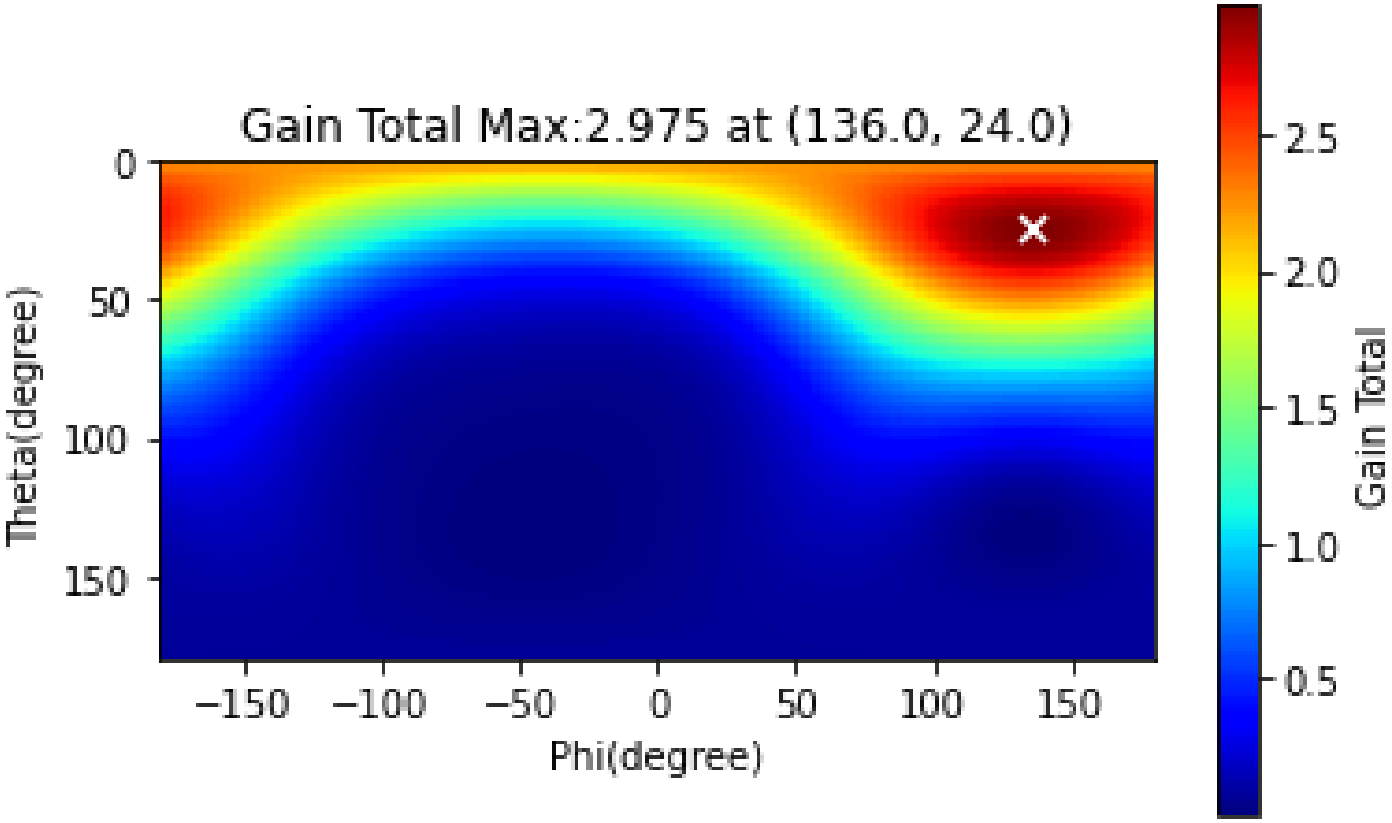
5. **數據視覺化** ：

- 使用 matplotlib 的 `imshow` 函數創建熱力圖，其中 x 軸為 phi，y 軸為 theta。
- 使用 `scatter` 函數標記出最大增益值的位置。
- 添加標籤、顏色條和標題，以提供更多信息。

6. **展示圖表** ：

- 使用 `plt.show()` 展示最終的圖表。

```
with open(r"C:\demo\Gain Table 1.csv") as f:
    text = f.readlines()

phi_set = set()
theta_set = set()
gain_total_list = []

for line in text[1:]:
    _, phi, theta, gain_total = map(float, line.split(','))
    gain_total_list.append(gain_total)
    phi_set.add(phi)
    theta_set.add(theta)

phi_list = sorted(list(phi_set))
theta_list = sorted(list(theta_set))

import numpy as np
z_2d = np.array(gain_total_list).reshape(len(phi_list), len(theta_list)).T
max_value = np.max(z_2d)
max_position = np.unravel_index(np.argmax(z_2d), z_2d.shape)
max_x, max_y = phi_list[max_position[1]], theta_list[max_position[0]]

import matplotlib.pyplot as plt
plt.imshow(z_2d, extent=(min(phi_list), max(phi_list), max(theta_list), min(the
plt.colorbar(label='Gain Total')
plt.scatter(max_x, max_y, color='white', s=50, marker='x')
plt.xlabel('Phi(degree)')
plt.ylabel('Theta(degree)')
plt.title(f'Gain Total Max:{max_value:.3f} at ({max_x}, {max_y})')
plt.show()
```



執行程式輸出圖片

數據提取

代碼片段是用來從 CSV 檔案中提取數據，並將這些數據整理為適合進行後續分析和視覺化的格式。下面是對代碼功能的詳細解釋：

1. **打開並讀取 CSV 檔案** ：

- 使用 `open` 函數以只讀模式打開位於指定路徑的 CSV 檔案。
- 使用 `readlines` 方法讀取檔案的所有行，每行作為列表 `text` 的一個元素。

2. **初始化集合和列表** ：

- `phi_set` 和 `theta_set` 集合用於儲存不重複的 phi 和 theta 值。
- `gain_total_list` 列表用於儲存所有的增益（gain_total）值。

3. **處理 CSV 檔案的每一行** ：

- 使用 for 迴圈遍歷 `text` 列表，從第二行開始（排除標題行）。
- 將每行分割成個別的值，然後轉換為浮點數。
- 將 phi 和 theta 值添加到對應的集合中，並將 gain_total 值添加到列表中。

4. **排序 phi 和 theta 值** ：

- 將 `phi_set` 和 `theta_set` 轉換為列表，並使用 `sorted` 函數進行排序。

至此，代碼成功地從 CSV 檔案中提取並整理了數據，為接下來的分析或圖形化處理做好了準備。

數據轉換及最大值查找

```
import numpy as np
z_2d = np.array(gain_total_list).reshape(len(phi_list), len(theta_list)).T
max_value = np.max(z_2d)
max_position = np.unravel_index(np.argmax(z_2d), z_2d.shape)
max_x, max_y = phi_list[max_position[1]], theta_list[max_position[0]]
```

代碼進行了以下幾個重要步驟，用於分析和處理從 CSV 檔案中提取的數據：

1. **創建二維數組** ：

- 使用 numpy 的 `array` 函數將 `gain_total_list` 轉換成 numpy 數組。
- 利用 `reshape` 方法將這個數組轉換成二維形式。這裡的形狀是由 `phi_list` 和 `theta_list` 的長度決定的，代表不同的 phi 和 theta 值的組合。
- 使用 `.T` （轉置操作）來調整數組的行和列，以便於後續的圖形化處理。

2. **尋找最大值及其位置** ：

- 使用 `np.max` 函數找到 `z_2d` 數組中的最大值，即 `max_value` 。
- 使用 `np.argmax` 結合 `np.unravel_index` 函數定位最大值在 `z_2d` 數組中的位置，即 `max_position` 。

3. **獲取最大值對應的 phi 和 theta 值** ：

- 根據 `max_position` 從 `phi_list` 和 `theta_list` 中提取相應的 phi 和 theta 值，這些值表示最大增益發生的位置。

這些步驟為您的數據分析提供了基礎，接下來可以利用這些信息進行圖形化展示或進一步的數據處理。

資料繪圖

```
import matplotlib.pyplot as plt
plt.imshow(z_2d, extent=(min(phi_list), max(phi_list), max(theta_list), min(the
plt.colorbar(label='Gain Total')
plt.scatter(max_x, max_y, color='white', s=50, marker='x')
plt.xlabel('Phi(degree)')
plt.ylabel('Theta(degree)')
plt.title(f'Gain Total Max:{max_value:.3f} at ({max_x}, {max_y})')
plt.show()
```

代碼現在包括了數據的視覺化部分，其中使用 matplotlib 庫來生成一張圖表。這段代碼的功能如下：

1. **繪製熱力圖** ：

- 使用 `plt.imshow` 函數繪製 `z_2d` 數組的熱力圖。這個數組包含了 phi 和 theta 對應的增益值。
- `extent` 參數用於設定 x 軸和 y 軸的範圍，這裡分別對應 phi 和 theta 的最小值和最大值。
- `interpolation='nearest'` 確保每個數據點都清晰可見。
- `cmap='jet'` 選擇了一個特定的顏色映射，以不同的顏色表示不同的增益值。

2. **添加顏色條** ：

- 使用 `plt.colorbar` 函數添加一個顏色條，以標示不同顏色對應的增益值。

3. **標記最大增益值的位置** ：

- 使用 `plt.scatter` 函數在最大增益值的位置標記一個白色的 "x"。
- `max_x` 和 `max_y` 確定了標記的位置。

4. **設置圖表的軸標籤和標題** ：

- 使用 `plt.xlabel` 和 `plt.ylabel` 分別設置 x 軸和 y 軸的標籤。
- 使用 `plt.title` 設置圖表的標題，其中包括最大增益值和其對應的 phi 和 theta 值。

5. **展示圖表** ：

- 使用 `plt.show` 函數來展示最終的圖表。

總結

使用 Python 進行模擬數據的後處理和視覺化提供了一種高效且自動化的方法來處理複雜的任務，如您所提到的尋找最大值。總結一下您的流程：

1. **數據匯出** ：首先，從 HFSS 或其他模擬工具中匯出數據為 CSV 格式，這種格式易於被 Python 讀取和處理。
2. **數據讀取和處理** ：通過 Python，您可以簡單地讀取 CSV 檔案，提取所需的數據（如 phi, theta 和增益值），並將它們轉換成適合分析的格式。
3. **數據分析** ：Python 提供的強大數據分析工具（如 numpy）使得尋找特定數據點（例如增益的最大值）成為可能，且過程自動化且高效。
4. **數據視覺化** ：使用 matplotlib 這類視覺化工具，您可以將數據以圖形的形式呈現，如通過熱力圖直觀地展示增益分佈，並在圖中標記出最大值的位置。
5. **自動化和重複利用** ：一旦設置好這個流程，您可以輕鬆地對新的數據集進行相同的處理，大大節省時間並提高工作效率。

總之，Python 在科學計算和數據視覺化方面的應用提供了一個強大的平台，可以幫助您更有效地處理和理解複雜的模擬數據。

4.2 Touchstone檔案處理

Touchstone 檔案是一種廣泛用於電子系統設計和模擬的數據檔案格式，尤其是在射頻/微波及高速訊號分析等工程領域。這種檔案格式主要用於儲存網絡參數，如 S-參數、Y-參數或 Z-參數等，這些參數是用來描述電子元件、電路和子系統的電氣行為。網路參數主要用於描述在一定頻率範圍內，電路元件對電磁波的反射和透射特性。它們是複數，表示振幅和相位信息。

在電子設計自動化（EDA）軟體中，如AEDT或HFSS等，Touchstone 檔案被廣泛用於射頻元件的模擬和分析。這些檔案可以從實際測量中獲得，也可以通過模擬來生成，以用於進一步的設計和分析工作。

Touchstone 檔案通常以 `.snp` 結尾，其中 `n` 代表檔案中包含的端口數量，例如 `.s2p` 是包含兩個端口的網絡參數檔案。這些檔案通常以文本格式儲存，可以用任何文本編輯器打開，內容包括頻率數據和對應的參數值。

檔案格式

- Touchstone 檔案格式是一種標準化的文本文件格式，用於儲存射頻（RF）元件的網絡參數，如 S-參數。以下是 Touchstone 檔案格式的主要特點：

1. **檔案擴展名** ：Touchstone 檔案通常以 `.snp` 結尾，其中 `n` 代表檔案包含的端口數量。例如，`.s2p` 表示兩個端口的參數。
2. **格式結構** ：

- **標題行** （可選）：以 `!` 開頭，用於提供註釋或描述性信息。
- **參數/單位定義行** ：以 `#` 開頭，定義頻率單位（如 GHz, MHz）、網絡參數類型（如 S, Y, Z 參數）和數據格式（如 RI = 實部/虛部，MA = 幅值/相位，DB = 分貝/相位）。
- **數據行** ：列出頻率和相應的網絡參數值。每行代表一個頻率點的數據。

3. **數據格式** ：

- **頻率值** ：根據標題行中定義的單位表示。
- **網絡參數** ：可以用不同的格式表示，如實部/虛部（RI）、幅值/相位（MA）、分貝/相位（DB）。每個參數都對應於特定的頻率值。

4. **多端口數據** ：在多端口檔案中（如 .s3p, .s4p），數據行將包含多個參數值，表示不同端口間的網絡參數。

```
! Touchstone file exported from Circuit Design 2022.2.0
!        File:           C:/Users/HFSS/Documents/Ansoft/Project15.aedt
!        Generated:      4:42:51 %x4E0B%x5348  %x4E5D%x6708 21, 2022
!        Design:         Circuit1
!        Project:        Project15
!        Setup:          LinearFrequency
!        Solution:       LinearFrequency
!
# GHz S MA R 50.000000
! Terminal data exported
! Port[1] = Port1
! Port[2] = Port2
! Port[3] = Port3
! Port[4] = Port4
0                       0.0148658752711880 -0               0.983264937451046 -0               0.000507409798865213 -0               0.000411139090159697 -180
                        0.982092941610736 -0                0.0172012548923387 -0              0.000417622136755297 -0               0.000494902868706938 -180
                        0.000505982061068827 -180             0.000420854500593506 -0              0.0161842529645679 -180             0.983523839999232 -0
                        0.000425059423550099 -0               0.000512903771025554 -0              0.983468561215064 -0                0.0178994064241289 -180
! Gamma         0          0                0                0                0                0                0                0
! Port Impedance         50                0                50               0                50               0                50               0
0.0145                  0.0321723100533284 52.3910900535703 0.982050902620007 -1.76215497016997 0.0240583430561527 85.936150117778  0.0208733813110567 -94.4450043437513
                        0.982038503833381 -1.733745227132  0.0319457765864731 54.0352244416086 0.0208710354709865 -94.9128502259872 0.0239733072988264 85.921508637157
                        0.024005880058122 85.8775949993308 0.0208509522987677 -94.7022813322782 0.0304518946856359 54.4882614691234 0.982896803167776 -1.63259946809827
                        0.0208171859123143 -94.6529413193703 0.0240562318907234 86.0105425450467 0.982736885026362 -1.68127579237517 0.0303338615821212 52.6168476065637
! Gamma         0          0                0                0                0                0                0                0
! Port Impedance         50                0                50               0                50               0                50               0
0.029                   0.056399000007019 65.272900999915  0.978275872374939 -3.39940722216211 0.0472167909880752 82.7536223826753 0.0410210029160432 -98.2439859021922
                        0.978542251440082 -3.4058485497005 0.0566140808444802 65.2364801181738 0.0408647354303414 -98.2663295807518 0.0473029130894449 82.7834251536986
                        0.047180238971748 82.7690754939203 0.0407691234212405 -98.2155719165642 0.0545221720288576 64.7810842547629 0.978725676956741 -3.27192793645231
                        0.0409019129270363 -98.2571810768468 0.0473578685193186 82.830138176988  0.978705175792622 -3.21807230092431 0.0546748605670967 65.2621786645783
! Gamma         0          0                0                0                0                0                0                0
! Port Impedance         50                0                50               0                50               0                50               0
0.0435                  0.0817753835464611 67.8002745592301 0.973422580186416 -5.01800994264676 0.0699545789666066 79.9559349243909 0.0603997585808573 -101.477716816852
                        0.974402522991419 -5.03474880917753 0.0796013769784463 68.0408403336199 0.0605401802101795 -101.27874278649 0.0700175207660075 80.132885108957
                        0.0697200141207571 79.9237095144215 0.0604001735249127 -101.226897151619 0.078257749990916 68.4601506410907 0.974522689860007 -4.81050574407324
                        0.0604252282895141 -101.503126607741 0.0699576704523781 80.1613728246477 0.974907745576576 -4.80508931547585 0.0765939734442163 66.8207588011664
! Gamma         0          0                0                0                0                0                0                0
! Port Impedance         50                0                50               0                50               0                50               0
0.058                   0.107083988272811 68.1096806573505 0.967860114577981 -6.59162845787935 0.0919850486270007 77.0591631285424 0.0794995077123656 -104.846672413002
                        0.967578472114537 -6.62282449707664 0.106033617527951 68.6125287919586 0.0795394596292879 -104.709433560786 0.0920012685760982 77.0536310676082
                        0.0918194191102797 77.1202982675426 0.0794787756965077 -104.674693672286 0.101160614480326 68.5214570984392 0.968329479686753 -6.32504078072408
                        0.0794585958246944 -104.925519432143 0.0920006489224473 77.0462639805282 0.968660396636387 -6.31522478734487 0.103352013062392 68.6874018415954
! Gamma         0          0                0                0                0                0                0                0
! Port Impedance         50                0                50               0                50               0                50               0
0.0725                  0.129488129680784 67.9971140134833 0.959802134866921 -8.10743865673312 0.113357305148236 74.4940151509937 0.0980254415735504 -107.727306936421
                        0.960051915034914 -8.13042919937612 0.130213315825929 67.8560320794554 0.0982208976031 6 -107.664281058159 0.113474338461751 74.5133261582496
                        0.113212559395886 74.540921096991  0.0979377147954766 -107.725688280042 0.123366454814377 67.9885797393188 0.961371195352563 -7.75254756937819
                        0.0981016981782705 -107.829789875476 0.113483322516147 74.4949805266054 0.960885107719359 -7.75076879638248 0.123813033373789 67.6622554798509
! Gamma         0          0                0                0                0                0                0                0
! Port Impedance         50                0                50               0                50               0                50               0
0.087                   0.153384162363145 65.8124407564    0.951732785265 91 -9.63057128607605 0.133477375677185 71 7655569895255 0.115337377900 67 -110.855851015
```

Touchstone檔案格式

檔頭資訊

字符串 `# GHz S MA R 50.000000` 是 Touchstone 檔案格式中的一個典型的參數/單位定義行。這行文字定義了該 Touchstone 檔案中數據的格式和單位。我將為您解釋這行文字的含義：

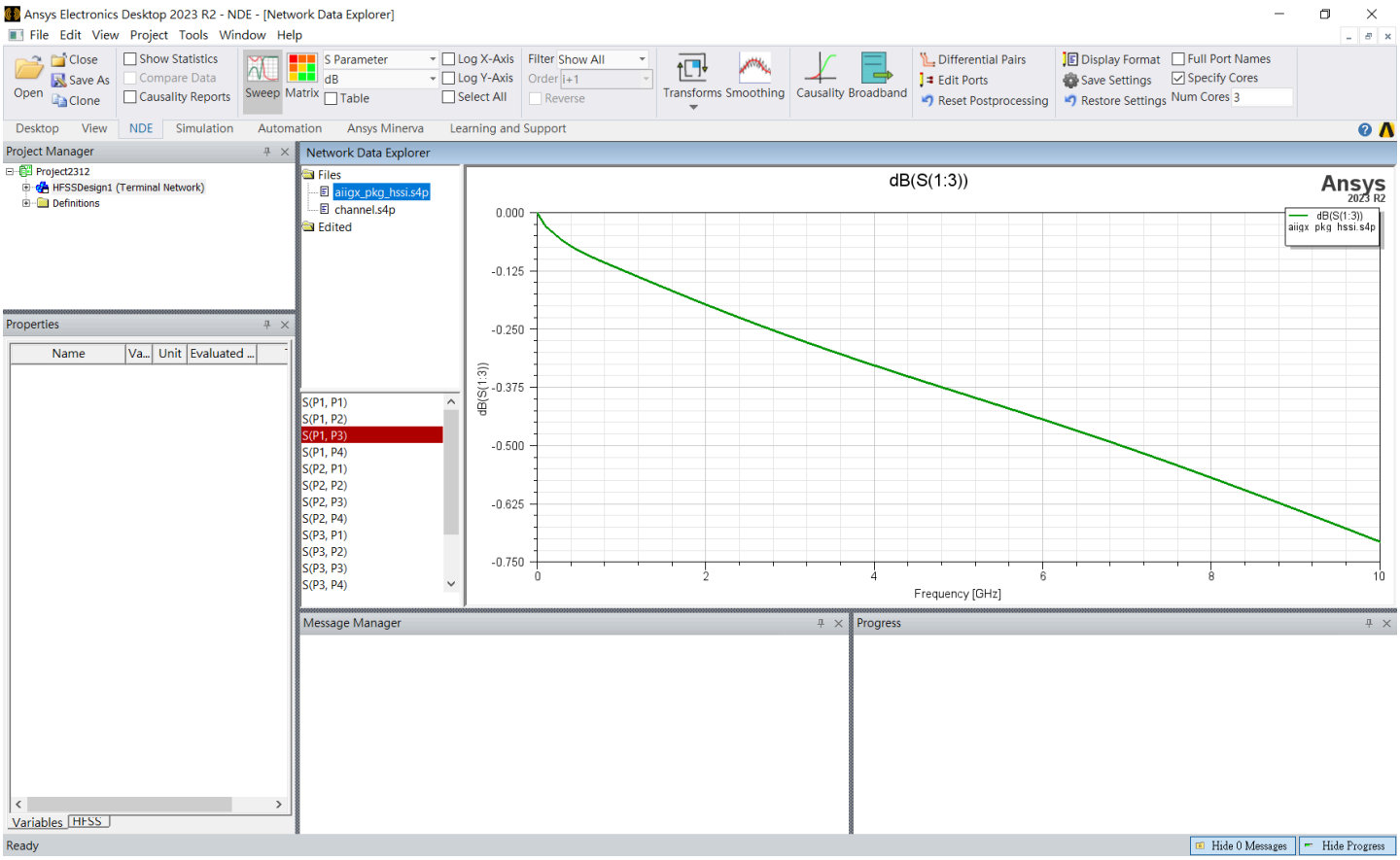
1. **#** ：這個符號表示這是一行設定或定義，而不是數據。
2. **GHz** ：這表示頻率單位是吉赫（GHz）。Touchstone 檔案中的頻率數據將以吉赫為單位。
3. **S** ：這表示檔案包含的是 S-參數（Scattering Parameters），它們是描述電子網絡（如放大器、濾波器等）在不同頻率下的反射和傳輸特性的參數。
4. **MA** ：這指數據的格式是幅值/相位（Magnitude-Angle）。這意味著每個 S-參數將以其幅值和相位的形式表示，而不是實部和虛部。
5. **R 50.000000** ：這表示參考阻抗是 50 歐姆。在 RF 和微波工程中，50 歐姆是一個非常常見的參考阻抗值。

因此，當讀取這個 Touchstone 檔案的數據時，您可以期望每行包含一個頻率值（以 GHz 為單位）和相應的 S-參數值（以幅值和相位的形式表示），並且這些數據是基於 50 歐姆的參考阻抗。

處理S參數檔案程式碼

Touchstone 檔案的檢視通常依靠專業工具進行，這些工具能夠將檔案匯入並在圖表中展示各種 S-參數曲線。在使用這些工具時，用戶需要逐一點擊想要查看的 S-參數曲線。然而，當涉及到多端口的情況，即端口數量較多時，這種方法可能變得相當繁瑣和時間消耗。每個端口都有其相應的 S-參數，這意味著用戶需要反覆地點擊不同的曲線來查看每個端口的特性。這不僅增加了操作的複雜性，也可能導致資料分析過程中的效率低下。

AEDT的 Network Data Explorer 可以用於高效地分析和可視化S參數數據。這個工具能夠直接讀取多種數據格式，包括 Touchstone 文件，從而簡化了多端口系統中複雜互連和元件的分析過程。



AEDT ND Explorer介面

即使 ND Explorer 是一款功能強大的工具，當面對特定情況，如處理大量的端口數或在圖表上加入標記（Marks）等細節時，它的便利性可能會有所降低。這主要是因為

這些複雜的操作要求更細致的數據管理和可視化設定。

在處理大量端口時，數據量的增加可能導致界面變得擁擠，使得特定數據點或趨勢線的識別和分析變得困難。此外，當需要在圖表上添加多個標記以突出特定的數據點或頻率時，操作可能變得繁瑣，尤其是在需要精準定位這些標記時。

為了解決這些問題，使用腳本可以幫助自動化一些繁瑣的任務，比如批量添加標記或管理大量的數據集。

複數資料型別

複數（Complex）是一種基本的資料型別，用來表示包含實部和虛部的數字。在複數中，實部和虛部都是浮點數，而虛部後面會加上 `j` 或 `J`。這種數字在數學中尤其在工程和物理領域中非常有用。

複數的表示

- 複數通常以 `a + bj` 的形式表示，其中 `a` 是實部，`b` 是虛部。
- 例如：`3 + 4j` 表示實部為 3，虛部為 4 的複數。

創建複數

- 直接賦值：`c = 3 + 4j`
- 使用 `complex` 函數：`c = complex(3, 4)`

複數的屬性

- `real`：獲取複數的實部。
- `imag`：獲取複數的虛部。

複數的操作

- 加法、減法、乘法和除法。
- 也可以使用一些內置的函數，比如 `abs()` 計算複數的模。

實例

```
c = 3 + 4j
print("實部：", c.real)
print("虛部：", c.imag)
print("模：", abs(c))
```

應用

複數在Python中主要用於科學計算，特別是在處理波動、振動或電路等領域的問題時。Python的標準庫和其他科學計算庫（如NumPy）提供了豐富的支持，使得複數的運算和應用非常方便。

總的來說，複數是Python中一個重要且有用的數據類型，對於需要進行複雜數學計算的領域尤為重要。

程式碼

以下的 Python 腳本是一個完整的流程，用於處理和可視化 Touchstone 檔案中的數據。這個腳本主要完成以下功能：

1. **定義一系列函數** ：

- `db_to_complex` 、 `ma_to_complex` 、 `ri_to_complex` 用於將不同格式的數據轉換為複數形式。
- `complex_to_db_phase` 用於將複數轉換為分貝值和相位。

2. **讀取 Touchstone 檔案** ：

- 從指定的路徑讀取 Touchstone 檔案。
- 解析檔案頭部的元數據，以確定數據的格式和端口數量。

3. **數據提取和轉換** ：

- 從檔案中提取數據並根據格式轉換為複數形式。
- 對每對端口組合計算 S 參數。

4. **數據可視化** ：

- 使用 matplotlib 繪製每對端口組合的 S 參數圖表。
- 為每個圖表設置標題、X 軸和 Y 軸標籤。

5. **保存圖表**：

- 將每個 S 參數曲線保存為單獨的 PNG 檔案。

完整代碼

```
# -*- coding: utf-8 -*-

import os
import re
import math
from itertools import product

def db_to_complex(db, angle_deg):
    if db < -100:
        db = -100

    if db > 100:
        db = 100
    magnitude = 10 ** (db / 20)
    angle_rad = math.radians(angle_deg)
    real = magnitude * math.cos(angle_rad)
    imaginary = magnitude * math.sin(angle_rad)

    return complex(real, imaginary)

def ma_to_complex(magnitude, angle_deg):
    angle_rad = math.radians(angle_deg)
    real = magnitude * math.cos(angle_rad)
    imaginary = magnitude * math.sin(angle_rad)

    return complex(real, imaginary)

def ri_to_complex(real, imaginary):

    return complex(real, imaginary)

def complex_to_db_phase(complex_number):
    magnitude = abs(complex_number)
    phase = math.degrees(math.atan2(complex_number.imag, complex_number.real))
    if magnitude == 0:
        db = float('1e-30')
```

```
    else:
        db = 20 * math.log10(magnitude)

    return db, phase

touchstone_path = 'c:/demo/channel.s4p'
extension = os.path.basename(touchstone_path).split('.')[1]
port_number = int(extension[1:-1])

with open(touchstone_path) as f:
    text = f.readlines()

data = []
ports = {}
for line in text:
    try:
        values = [float(i) for i in line.strip().split()]
        data += values

    except:
        if line.startswith('#'):
            _, unit, network_type, data_format, _, Z = line.split()
            if data_format == 'RI':
                convert = ri_to_complex
            elif data_format == 'DB':
                convert = db_to_complex
            elif data_format == 'MA':
                convert = ma_to_complex

        else:
            m = re.search('!\sPort\[(\d+)\]\s=\s(.*)$', line)
            if m:
                num = int(m.group(1))
                name = m.group(2)
                ports[num] = name

frequencies = data[::2*port_number**2+1]

matrix = {}
```

```
for m, n in product(range(port_number), range(port_number)):
    vm = data[(m*2*port_number+n*2+1)::(2*port_number**2 + 1)]
    vn = data[(m*2*port_number+n*2+2)::(2*port_number**2 + 1)]
    matrix[(m+1, n+1)] = [convert(i, j) for i, j in zip(vm, vn)]

import matplotlib.pyplot as plt
for m, n in product(range(port_number), range(port_number)):
    x = frequencies
    y = [complex_to_db_phase(v)[0] for v in matrix[(m+1, n+1)]]
    plt.plot(x, y)
    plt.grid()
    plt.title(f'S({m+1},{n+1})')
    plt.xlabel(f'Freq ({unit})')
    plt.ylabel('dB')
    plt.savefig(f'c:/demo/S_{m+1}_{n+1}.png')
    plt.clf()
```

程式輸出所有S參數圖片

導入模組說明

```
# -*- coding: utf-8 -*-

import os
import re
import math
from itertools import product
```

這段 Python 代碼是一個基本的輸入，用於設置編碼和導入一些常用的庫和模塊。這裡是每個導入部分的簡要說明：

1. `# -*- coding: utf-8 -*-`：這行告訴 Python 解釋器，該文件使用 UTF-8 編碼。這是一個好習慣，尤其是當您的代碼中包含非ASCII字符時。
2. `import os`：導入 `os` 模塊，它提供了一種方便的方法來使用操作系統相關的功能，如文件和目錄的處理。
3. `import re`：導入 `re` 模塊，它支持正則表達式的操作，這在文本處理中非常有用。
4. `import math`：導入 `math` 模塊，它提供了一套廣泛的數學函數，包括對數、三角函數等。
5. `from itertools import product`：從 `itertools` 模塊導入 `product` 函數。`itertools` 是 Python 的一個標準庫，專門用於高效迭代操作。`product` 函數是用於生成笛卡爾積的。

轉換函數定義

```
def db_to_complex(db, angle_deg):
    if db < -100:
        db = -100

    if db > 100:
        db = 100
    magnitude = 10 ** (db / 20)
    angle_rad = math.radians(angle_deg)
    real = magnitude * math.cos(angle_rad)
    imaginary = magnitude * math.sin(angle_rad)

    return complex(real, imaginary)

def ma_to_complex(magnitude, angle_deg):
    angle_rad = math.radians(angle_deg)
    real = magnitude * math.cos(angle_rad)
    imaginary = magnitude * math.sin(angle_rad)

    return complex(real, imaginary)

def ri_to_complex(real, imaginary):

    return complex(real, imaginary)

def complex_to_db_phase(complex_number):
    magnitude = abs(complex_number)
    phase = math.degrees(math.atan2(complex_number.imag, complex_number.real))
    if magnitude == 0:
        db = float('1e-30')
    else:
        db = 20 * math.log10(magnitude)

    return db, phase
```

代碼定義了四個函數，用於執行不同類型的數據轉換，這些函數對於處理和分析複數信號非常有用。以下是對每個函數的簡要解釋：

1. `db_to_complex(db, angle_deg)`：這個函數將分貝值（dB）和角度（以度為單位）轉換為複數。函數首先將分貝值限制在 -100 到 100 的範圍內，然後根據分貝值計算幅值，並將角度轉換為弧度，最後使用三角函數計算實部和虛部。
2. `ma_to_complex(magnitude, angle_deg)`：這個函數將幅值和角度轉換為複數。與 `db_to_complex` 類似，它使用三角函數來計算實部和虛部，但是這裡直接使用提供的幅值。
3. `ri_to_complex(real, imaginary)`：這是一個簡單的函數，直接根據給定的實部和虛部值創建一個複數。
4. `complex_to_db_phase(complex_number)`：這個函數將複數轉換為分貝值和相位。首先計算複數的模（幅值），然後計算相位角。如果模為零，則分貝值設定為一個極小的正數（以避免對數運算中的數學錯誤）；否則，使用對數函數計算分貝值。

> dB（分貝）是一種用於描述聲音強度或者其他信號（如電信號）相對於參考值的對數單位。它廣泛應用於聲學、電子學、通訊工程等領域中，用來量化信號的強度或功率水平。分貝基於對數比例來表示值，這使得它特別適合於表示範圍很廣的量。例如，人耳可聽範圍的聲壓級就涵蓋了極大的範圍，使用分貝可以更方便地表述這些值。

設定路徑

```
touchstone_path = 'c:/demo/channel.s4p'
extension = os.path.basename(touchstone_path).split('.')[1]
port_number= int(extension[1:-1])
```

這段代碼是用來從 Touchstone 檔案的路徑中提取有關該檔案的一些基本信息。具體來說，它確定了檔案的端口數量。下面是代碼的逐行解釋：

1. `touchstone_path = 'c:/demo/channel.s4p'`：這一行定義了 Touchstone 檔案的路徑。在這個例子中，檔案名為 `channel.s4p`，位於 `c:/demo/` 目錄下。
2. `extension = os.path.basename(touchstone_path).split('.')[1]`：這一行提取了檔案的擴展名。`os.path.basename` 函數獲取檔案的基本名稱（即不包含路徑的部

分），然後通過 `split` 方法在點（`.`）處分割字符串，取第二個元素（索引為 1 的元素），即檔案的擴展名。

3. `port_number= int(extension[1:-1])`：這一行從擴展名中提取端口數量。Touchstone 檔案的命名規則通常是 `.snp`，其中 `n` 表示端口的數量。代碼通過 `int(extension[1:-1])` 從擴展名中取出 `n` 的值並將其轉換為整數。

檔案內容提取

```
with open(touchstone_path) as f:
    text = f.readlines()

data = []
ports = {}
for line in text:
    try:
        values = [float(i) for i in line.strip().split()]
        data += values

    except:
        if line.startswith('#'):
            _, unit, network_type, data_format, _, Z = line.split()
            if data_format == 'RI':
                convert = ri_to_complex
            elif data_format == 'DB':
                convert = db_to_complex
            elif data_format == 'MA':
                convert = ma_to_complex

        else:
            m = re.search('!\sPort\[(\d+)\]\s=\s(.*)$', line)
            if m:
                num = int(m.group(1))
                name = m.group(2)
                ports[num] = name
```

這段代碼是用來讀取 Touchstone 檔案並從中提取重要的信息。以下是代碼的逐行解釋：

1. **讀取檔案** ：

- `with open(touchstone_path) as f:` ：使用 `with` 語句打開檔案，這樣可以確保檔案在讀取後被正確關閉。
- `text = f.readlines()` ：讀取檔案的所有行並將它們儲存在 `text` 變量中。

2. **初始化變量** ：

- `data = []` ：創建一個空列表來儲存數據值。
- `ports = {}` ：創建一個空字典來儲存端口信息。

3. **解析檔案內容** ：

- 使用 `for` 循環遍歷檔案的每一行。
- 將每行的內容分割並嘗試將其轉換為浮點數，然後添加到 `data` 列表中。
- 如果轉換失敗（例如，當處理非數據行時），則處理異常。

4. **處理特殊行** ：

- 如果行以 `#` 開頭，則從中解析單位、網絡類型、數據格式和參考阻抗值。根據數據格式（ `RI` 、 `DB` 或 `MA` ），設置相應的轉換函數。
- 如果行包含端口描述（如 `! Port[1] = ...` ），則使用正則表達式 `re.search` 從中提取端口號和描述。

這段代碼為後續處理 Touchstone 檔案的數據提供了必要的前置步驟，包括數據的讀取和初步的解析，以及根據檔案的格式設定適當的數據轉換函數。通過這種方法，您可以準備好數據以進行進一步的處理和分析。

zip與拆包

在Python中， `zip` 函數和拆包（unpacking）是兩種常用的功能，它們在處理序列和迭代器時非常有用。

Zip 函數

`zip` 函數可以將多個可迭代物件（如列表、元組、字典等）組合成一個迭代器。它將這些可迭代物件中對應位置的元素配對，形成一系列的元組。

示例：

假設有兩個列表：

```
a = [1, 2, 3]
b = ['a', 'b', 'c']
```

使用 `zip(a, b)` 得到的結果會是一個迭代器，其中包含元組：`(1, 'a')`, `(2, 'b')`, `(3, 'c')`。

拆包（Unpacking）

拆包是指將序列中的元素解包到多個變數中。在Python中，您可以輕鬆地從列表、元組或任何可迭代物件中解包元素。

示例：

```
x, y, z = [1, 2, 3]
```

這裡，列表 `[1, 2, 3]` 中的元素分別被賦值給變數 `x`, `y`, `z`。

zip 和 拆包 結合使用

`zip` 和拆包可以結合使用來處理多個序列。例如，您可以使用 `zip` 將多個列表的元素配對，然後通過拆包將配對的元素分配給不同的變數。

示例：

```
for a, b in zip([1, 2, 3], ['a', 'b', 'c']):
    print(a, b)
```

這裡，`zip` 將兩個列表中的元素配對，然後在 for 循環中，每一對元素被拆包到變數 `a` 和 `b`。

這些技巧在數據處理和迭代操作中非常有用，可以讓代碼更加簡潔和可讀。

提取頻率信息和組成網絡參數矩陣

```
frequencies = data[::2*port_numner**2+1]

matrix = {}
for m, n in product(range(port_numner), range(port_numner)):
    vm = data[(m*2*port_numner+n*2+1)::(2*port_numner**2 + 1)]
    vn = data[(m*2*port_numner+n*2+2)::(2*port_numner**2 + 1)]
    matrix[(m+1, n+1)] = [convert(i, j) for i, j in zip(vm, vn)]
```

這段代碼用於從讀取的 Touchstone 檔案數據中提取頻率信息和組成網絡參數矩陣。以下是對代碼的逐行解釋：

1. **提取頻率數據** ：

- `frequencies = data[::2*port_numner**2+1]`：這行從 `data` 列表中提取頻率值。頻率值是按照一定間隔分佈的，間隔取決於端口數量的平方（因為 S-參數矩陣是一個平方矩陣）。

> 在 Touchstone 文件中，每個頻率點會有n*n組複數(每組2筆數值)數據，其中 n 是端口的數量。如果這些數據前面跟隨著頻率點本身的數值，那麼每個頻率點的總數據量會是:
>
> $$2 \times n^2 + 1$$
>
> 在 Python 中，您可以使用切片操作 `[::2*n**2+1]` 來提取出所有頻率值。當n=4時，每隔33筆數值會進到下一個頻率區段。

2. **構建網絡參數矩陣** ：

- 初始化一個空字典 `matrix` 來存儲網絡參數。
- 使用 `product(range(port_numner), range(port_numner))` 生成端口對的組合。
- 對於每一對端口 `(m, n)`，從 `data` 中提取對應的 S-參數數據。
- `vm` 和 `vn` 是通過特定的索引間隔從 `data` 中提取的數據。這些索引是根據 Touchstone 檔案的數據布局計算得出的。
- 使用 `zip(vm, vn)` 結合相應的數據點，並將它們通過之前確定的轉換函數（`convert`）轉換為複數形式，儲存在 `matrix` 字典中。這裡用到zip+拆包的技

巧。

綜上所述，這段代碼將 Touchstone 檔案中的原始數據轉換為一個組織化的矩陣格式，每個元素代表特定端口組合的 S-參數值。這為後續分析這些參數提供了基礎。

> 當 `port_number` 等於 4 時，使用 `product(range(port_number), range(port_number))` 生成的所有 `(m, n)` 組合如下：
>
> (0, 0), (0, 1), (0, 2), (0, 3)
> (1, 0), (1, 1), (1, 2), (1, 3)
> (2, 0), (2, 1), (2, 2), (2, 3)
> (3, 0), (3, 1), (3, 2), (3, 3)
>
> 這裡的每一對 `(m, n)` 都代表一個特定的端口組合，總共有 16 種組合（4x4）。

繪圖並輸出

```
import matplotlib.pyplot as plt
for m, n in product(range(port_numner), range(port_numner)):
    x = frequencies
    y = [complex_to_db_phase(v)[0] for v in matrix[(m+1, n+1)]]
    plt.plot(x, y)
    plt.grid()
    plt.title(f'S({m+1},{n+1})')
    plt.xlabel(f'Freq ({unit})')
    plt.ylabel('dB')
    plt.savefig(f'c:/demo/S_{m+1}_{n+1}.png')
    plt.clf()
```

代碼是用於繪製並儲存一系列頻率響應圖的Python程式。這裡使用了`matplotlib`庫來繪製圖表。您的代碼的目標是對於每一對`m`和`n`（這些來自`port_number`的範圍），繪製一個相應的頻率響應圖。每張圖表顯示的是從矩陣`matrix[(m+1, n+1)]`中提取的數據，並將其轉換為分貝相位。

以下是代碼的幾個要點：

1. 使用`product`函數來獲得所有可能的`m`和`n`組合。
2. 對於每對`m`和`n`，利用`frequencies`數組和`matrix`中的數據繪製一個圖表。
3. 圖表以分貝（dB）為單位顯示相位響應。
4. 每個圖表被儲存為一個PNG文件，文件名包含了對應的`m`和`n`值。