Tuyệt vời! Việc bạn đã nắm vững ý tưởng khái quát là bước quan trọng nhất. Bây giờ, chúng ta sẽ cùng nhau "xây dựng" lại quy trình tối ưu hóa của bạn bằng Tối ưu hóa Bayes, đi sâu vào từng thành phần và hành động cụ thể.

Hãy tưởng tượng chúng ta đang thay thế cỗ máy "Grid Search" cồng kềnh bằng một "Robot thông minh" (BO). Đây là cách chúng ta lắp ráp và hướng dẫn robot đó.

### Bước 1: Xây dựng "Hộp đen" - Hàm mục tiêu (Objective Function)

Đây là bước nền tảng. Bạn cần gói gọn toàn bộ quá trình tính toán chi phí của mình vào một hàm Python duy nhất. Robot BO không cần biết bên trong có gì, nó chỉ cần biết cách "nhấn nút" và nhận kết quả.

**Nhiệm vụ của hàm này:**

* **Đầu vào:** Nhận một danh sách (list) hoặc mảng (array) chứa 8 giá trị tham số theta (ví dụ: [theta\_L1, theta\_L2,..., theta\_C4]).
* **Đầu ra:** Trả về một con số duy nhất (một float) là giá trị hàm chi phí Lambda(m).

**Cách thực hiện:**

Bạn sẽ kết hợp logic từ các file notebook của mình:

1. Lấy 8 giá trị theta làm đầu vào.
2. Sử dụng logic từ Zin1.ipynb để tính ma trận ABCD\_total tại tần số mục tiêu (1.3 GHz).
3. Từ ABCD\_total và trở kháng tải Z\_load (50 Ohm), tính ra trở kháng vào Z\_in.1
4. Sử dụng logic từ update\_class\_nearest\_point.ipynb:
   * Chuyển Z\_in thành tọa độ trên biểu đồ Smith.
   * Tính khoảng cách ngắn nhất d\_in từ điểm Z\_in này đến đường bao mục tiêu (đã được tạo sẵn cho Pin và tần số 1.3 GHz).
5. Tính chi phí: cost = d\_in\*\*2.1
6. Trả về cost.

**Mã nguồn khái niệm:**

Python

# Giả sử bạn đã có đối tượng 'target\_contour' được tạo sẵn cho Pin và tần số 1.3 GHz  
# target\_contour = prepare\_target\_contour(full\_df, pin\_value=18, freq\_ghz=1.3)  
  
def objective\_function(params):  
 """  
 Hàm mục tiêu "hộp đen" cho Bayesian Optimization.  
   
 Args:  
 params (list): Một danh sách 8 giá trị [theta\_L1,..., theta\_C4].  
   
 Returns:  
 float: Giá trị hàm chi phí.  
 """  
 # Giải nén 8 tham số  
 theta\_L1, theta\_L2, theta\_L3, theta\_L4, \  
 theta\_C1, theta\_C2, theta\_C3, theta\_C4 = params  
  
 # --- BƯỚC 1: TÍNH Z\_IN (Logic từ Zin1.ipynb) ---  
 #... (toàn bộ code tính ma trận ABCD và Z\_in với 8 tham số trên)  
 # zin\_complex = calculate\_zin(...)   
   
 # --- BƯỚC 2: TÍNH CHI PHÍ (Logic từ update\_class\_nearest\_point.ipynb) ---  
 # Chuyển Zin sang hệ số phản xạ Gamma  
 # test\_point\_gamma =...  
   
 # Tìm khoảng cách đến đường bao mục tiêu  
 # distance = target\_contour.find\_nearest\_point\_on\_envelope(test\_point\_gamma)  
   
 # Tính chi phí  
 cost = distance\*\*2  
   
 print(f"Đang thử bộ tham số: {[round(p, 2) for p in params]} -> Chi phí: {cost:.6f}")  
 return cost

### Bước 2: Định nghĩa "Sân chơi" - Không gian tìm kiếm (Search Space)

Robot BO cần biết giới hạn của sân chơi. Bạn phải định nghĩa rõ ràng phạm vi cho phép của từng tham số trong 8 tham số của mình.

**Cách thực hiện:**

Trong Python, đây thường là một danh sách các tuple, mỗi tuple chứa giá trị tối thiểu và tối đa cho một tham số.

**Mã nguồn khái niệm:**

Python

# Mỗi tham số theta có thể chạy từ 0 đến 180 độ  
dimensions = [  
 (0.0, 180.0), # theta\_L1  
 (0.0, 180.0), # theta\_L2  
 (0.0, 180.0), # theta\_L3  
 (0.0, 180.0), # theta\_L4  
 (0.0, 180.0), # theta\_C1  
 (0.0, 180.0), # theta\_C2  
 (0.0, 180.0), # theta\_C3  
 (0.0, 180.0) # theta\_C4  
]

### Bước 3: Lắp ráp "Bộ não" của Robot - Các thành phần cốt lõi của BO

Đây là phần lý thuyết quan trọng nhất, giải thích cách robot "suy nghĩ".

#### 3.1. Mô hình Thay thế (Surrogate Model): "Tấm bản đồ địa hình xác suất"

* **Nó là gì?** Như đã nói, đây là một mô hình thống kê (thường là Quy trình Gauss - GP) dùng để xấp xỉ hàm chi phí "đắt đỏ" của bạn.
* **Nó hoạt động thế nào?** Sau mỗi lần chạy objective\_function, GP sẽ cập nhật "tấm bản đồ" của nó. Tấm bản đồ này cho biết 2 điều tại mọi điểm trong không gian 8 chiều:
  1. **Dự đoán (Mean):** "Tôi dự đoán chi phí ở đây là X."
  2. **Độ không chắc chắn (Uncertainty):** "Nhưng tôi không chắc lắm, giá trị thực có thể dao động trong khoảng này."
* **Tại sao nó hữu ích?** Bản đồ này cực kỳ rẻ để tính toán. Thay vì phải chạy hàm mục tiêu tốn kém, robot có thể "nhìn" vào bản đồ này để đưa ra quyết định.

#### 3.2. Hàm Thu thập (Acquisition Function): "Người cố vấn chiến lược"

* **Nó là gì?** Đây là một hàm toán học giúp robot quyết định điểm tiếp theo cần thử nghiệm. Nó nhìn vào "bản đồ" của Mô hình Thay thế và tính toán một "điểm số hấp dẫn" cho mọi điểm chưa được thử.
* **Nó hoạt động thế nào?** Một hàm thu thập phổ biến là **"Cải thiện Kỳ vọng" (Expected Improvement - EI)**. Hàm này tính toán điểm số cao cho những nơi:
  + Có chi phí dự đoán **rất thấp** (chiến lược **Khai thác**).
  + VÀ/HOẶC có độ không chắc chắn **rất cao** (chiến lược **Khám phá**).
* **Tại sao nó hữu ích?** Sự cân bằng giữa Khai thác và Khám phá giúp thuật toán không bị "tham lam" đi vào một cực tiểu cục bộ. Nó vừa cố gắng tinh chỉnh giải pháp tốt nhất hiện tại, vừa dành thời gian để khám phá những vùng hoàn toàn mới lạ, vì biết đâu ở đó có một giải pháp tốt hơn nhiều đang ẩn giấu.1

### Bước 4: Triển khai thực tế với Thư viện Python

Bạn không cần phải tự viết lại toàn bộ logic của BO. Có những thư viện tuyệt vời đã làm sẵn cho bạn. Một trong những thư viện phổ biến và dễ sử dụng nhất là scikit-optimize.

**Cách thực hiện:**

1. **Cài đặt thư viện:**  
   Bash  
   pip install scikit-optimize
2. Sử dụng hàm gp\_minimize:  
   Hàm này sẽ gói gọn toàn bộ vòng lặp tối ưu hóa cho bạn.

**Mã nguồn hoàn chỉnh (kết hợp tất cả các bước):**

Python

import numpy as np  
from skopt import gp\_minimize  
from skopt.space import Real  
from skopt.utils import use\_named\_args  
  
# --- BƯỚC 2: ĐỊNH NGHĨA KHÔNG GIAN TÌM KIẾM ---  
# Sử dụng đối tượng của scikit-optimize để dễ quản lý hơn  
dimensions =  
  
# --- BƯỚC 1: XÂY DỰNG HÀM MỤC TIÊU ---  
# (Hàm objective\_function đã định nghĩa ở trên)  
#...  
  
# --- BƯỚC 4: CHẠY THUẬT TOÁN TỐI ƯU HÓA ---  
  
# Số lần gọi hàm mục tiêu (ví dụ: 200 lần)  
# Thay vì ~1.7 triệu lần của Grid Search, ta chỉ cần vài trăm lần.  
N\_CALLS = 200   
  
print("Bắt đầu quá trình Tối ưu hóa Bayes...")  
  
# Gọi hàm chính của scikit-optimize  
result = gp\_minimize(  
 func=objective\_function, # Hàm mục tiêu "đắt đỏ" của bạn  
 dimensions=dimensions, # Không gian tìm kiếm  
 acq\_func="EI", # Hàm thu thập: Expected Improvement  
 n\_calls=N\_CALLS, # Tổng số lần chạy  
 n\_initial\_points=10, # Số điểm khởi tạo ngẫu nhiên  
 random\_state=123 # Để kết quả có thể tái lặp  
)  
  
# --- BƯỚC 5: XEM KẾT QUẢ ---  
  
print("\nQuá trình tối ưu hóa hoàn tất!")  
print(f"Chi phí tốt nhất tìm được: {result.fun:.6f}")  
print("Bộ tham số tối ưu:")  
best\_params = result.x  
param\_names = [d.name for d in dimensions]  
for name, value in zip(param\_names, best\_params):  
 print(f" {name}: {value:.4f}")  
  
# Bạn cũng có thể trực quan hóa quá trình hội tụ  
from skopt.plots import plot\_convergence  
plot\_convergence(result)

### Bước 5: Phân tích kết quả

Sau khi chạy xong, result sẽ chứa mọi thứ bạn cần:

* result.fun: Giá trị chi phí nhỏ nhất mà thuật toán tìm được.
* result.x: Danh sách 8 giá trị tham số theta tương ứng với chi phí nhỏ nhất đó.
* plot\_convergence(result): Sẽ vẽ một biểu đồ cho thấy chi phí tốt nhất giảm dần qua mỗi lần lặp. Nếu đường cong đi ngang ở cuối, có nghĩa là thuật toán đã hội tụ và khó có thể tìm ra giải pháp tốt hơn nữa.

Bằng cách thực hiện theo các bước này, bạn đã thay thế hoàn toàn phương pháp tìm kiếm vét cạn bằng một kỹ thuật tối ưu hóa toàn cục hiện đại và hiệu quả hơn rất nhiều.1 Nó không chỉ tiết kiệm thời gian tính toán mà còn tăng khả năng tìm ra một giải pháp thực sự tốt.