Powering Innovation That Drives Human Advancement

PyOptiSLang工作坊

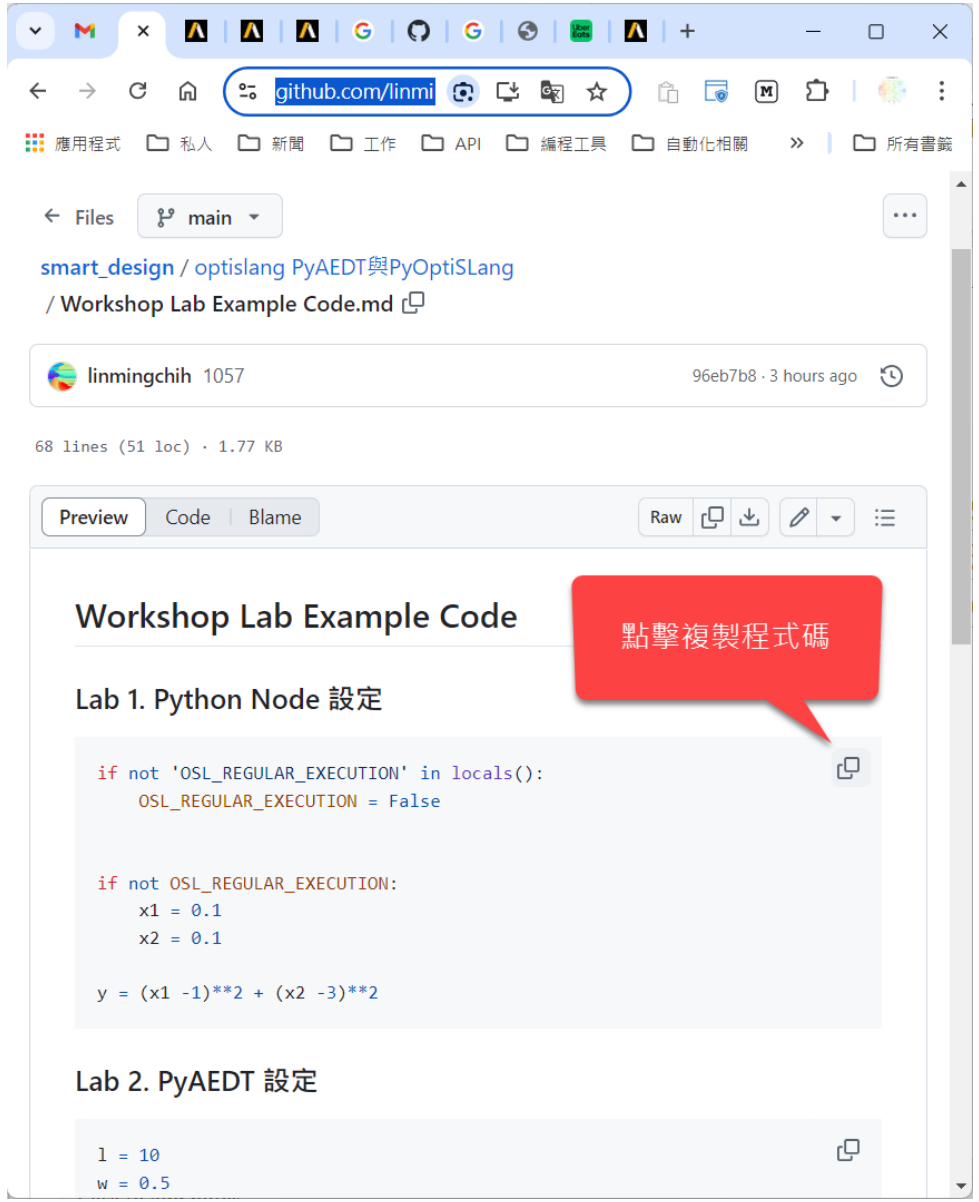
講師：林鳴志

Agenda

- 9:30 - 10:40
  - PyAEDT與PyOptiSLang安裝
  - 在optiSLang當中運行Python程式碼(Lab 1)
  - 運行PyAEDT程式碼(Lab 2)
- 10:50 - 12:00
  - 將pyaedt程式碼嵌入optiSLang (Lab3)
  - PyOptiSLang生成模型與優化 (Lab4, 5)

Workshop範例代碼複製

- 範例代碼連結
  - https://github.com/linmingchih/smart_design/blob/main/optislang%20PyAEDT%E8%88%87PyOptiSLang/Workshop%20Lab%20Example%20Code.md



建立虛擬Python虛擬環境與模組安裝

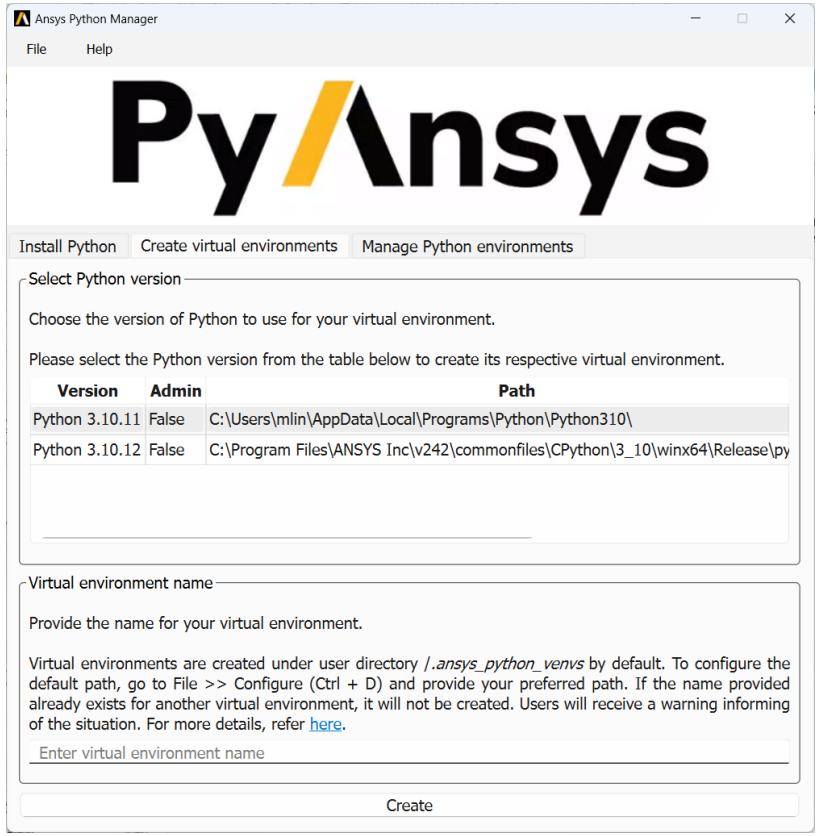
- Python 虛擬環境
  - Python 專案建立獨立的環境。透過虛擬環境，專案可以使用不同版本的 Python 解釋器和依賴套件，避免與全局環境發生衝突。venv 是內建於 Python 的模組，使用指令 python -m venv myenv 創建虛擬環境，啟動環境則使用 source myenv/bin/activate (Linux/macOS) 或 myenv\Scripts\activate (Windows)。
- PyAEDT
  - PyAEDT 是 Ansys Electronics Desktop 的 Python API，允許使用者透過腳本自動化電子設計軟體（如 HFSS、Maxwell 和 Q3D 等）的操作。它提供了一個高層次的介面，能夠簡化設計過程中的模擬設置、結果分析和優化步驟，特別適合電磁場、熱分析及多物理場耦合模擬。PyAEDT 支援無頭模式運行，有助於節省計算資源，並能輕鬆集成到其他 Python 工具或流程中，進行大規模的批次處理與自動化。
- PyOptiSLang
  - PyOptislang 是 optiSLang 的 Python API，專為優化、敏感性分析及設計自動化而設計。透過 PyOptislang，可以無縫地整合優化工具到設計流程中，利用其強大的算法來尋找最佳解。它支援多種設計優化任務，包括參數探索、多目標優化及機器學習模型建模。PyOptislang 能與各種仿真軟體合作，並自動化優化過程中的任務分配與數據處理，極大提升設計效率。

建立虛擬環境與PyAEDT/PyOptiSLang安裝

- 命令利用指令建立myvenv並安裝模組

```
rmdir /S /Q c:\myvenv
cd C:\Program
Files\AnsysEM\v241\Win64\commonfiles\CPython\
3_10\winx64\Release\python
.\python -m venv c:\myvenv
cd c:\myvenv\Scripts
activate
pip install pyaedt
pip install ansys-optislang-core
```

- 利用PyANSYS安裝



1. 在optiSLang當中運行Python程式碼

Python Node

- 在 optiSLang 中，你可以選擇不同的 Python 環境來執行腳本。預設的 Python 環境是 optiSLang Python。下面是一些關於 Python 環境設定的詳細說明：
  - 預設環境: optiSLang 內建的 Python 環境預設為 64 位元版本，這是為了支援共享庫（如 .dll 或 .so）的要求。這個環境已經包含了一些基礎的 Python 模組，但如果需要額外的套件，你可以自行安裝並擴充這個環境。
  - 選擇其他環境: 除了 optiSLang Python 外，你也可以選擇系統中其他已安裝的 Python 環境，例如 Python 3.5 64 bit 或 Anaconda 3.5 64 bit。這些環境可以提供更靈活的工具和套件，特別是在資料科學或機器學習應用上。

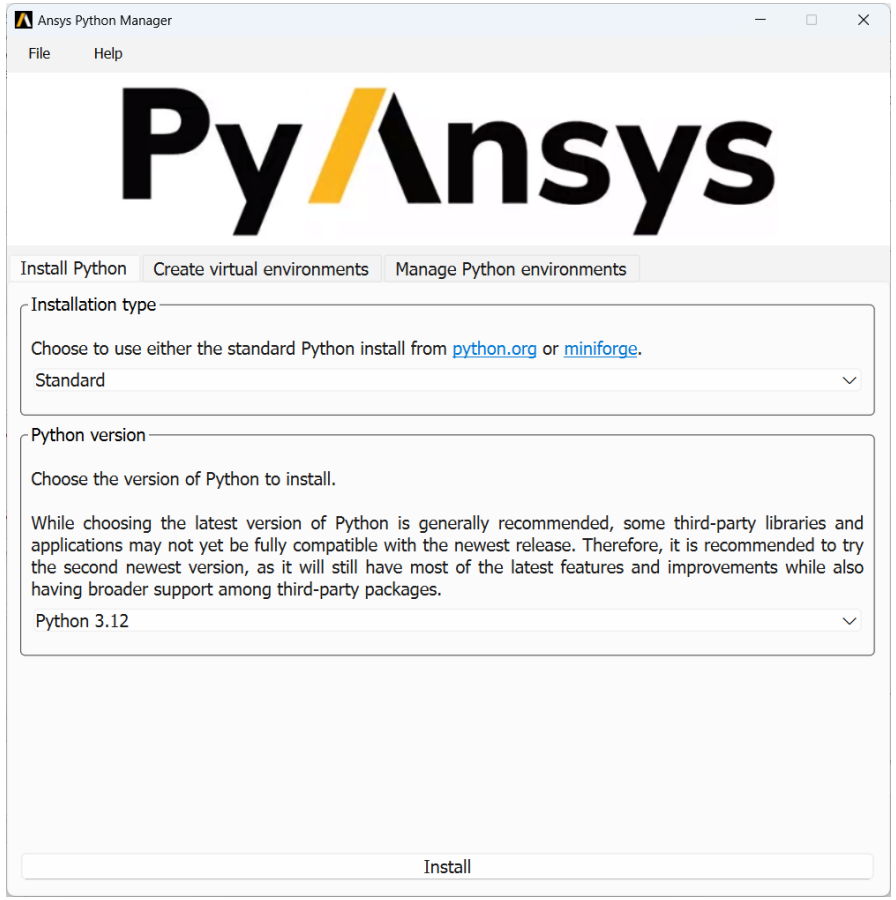
Lab 1. 熟悉Python節點設定

- 新增參數化系統節點（Parametric System node）。
- 在參數化系統節點中新增Python 節點（Python node）。
- 將程式碼複製到Python 節點中。
- 測試運行程式碼。
- 將 'x1'、'x2' 設定為參數（Parameter），將 'y' 設定為響應（Response）。
- 在參數化系統中設定 'x1' 和 'x2' 的範圍。
- 將靈敏度向導（Sensitivity Wizard）拖曳到參數化系統節點的標題上。
- 點擊 Next > 完成設置。
- 執行計算。

2. 運行PyAEDT程式碼

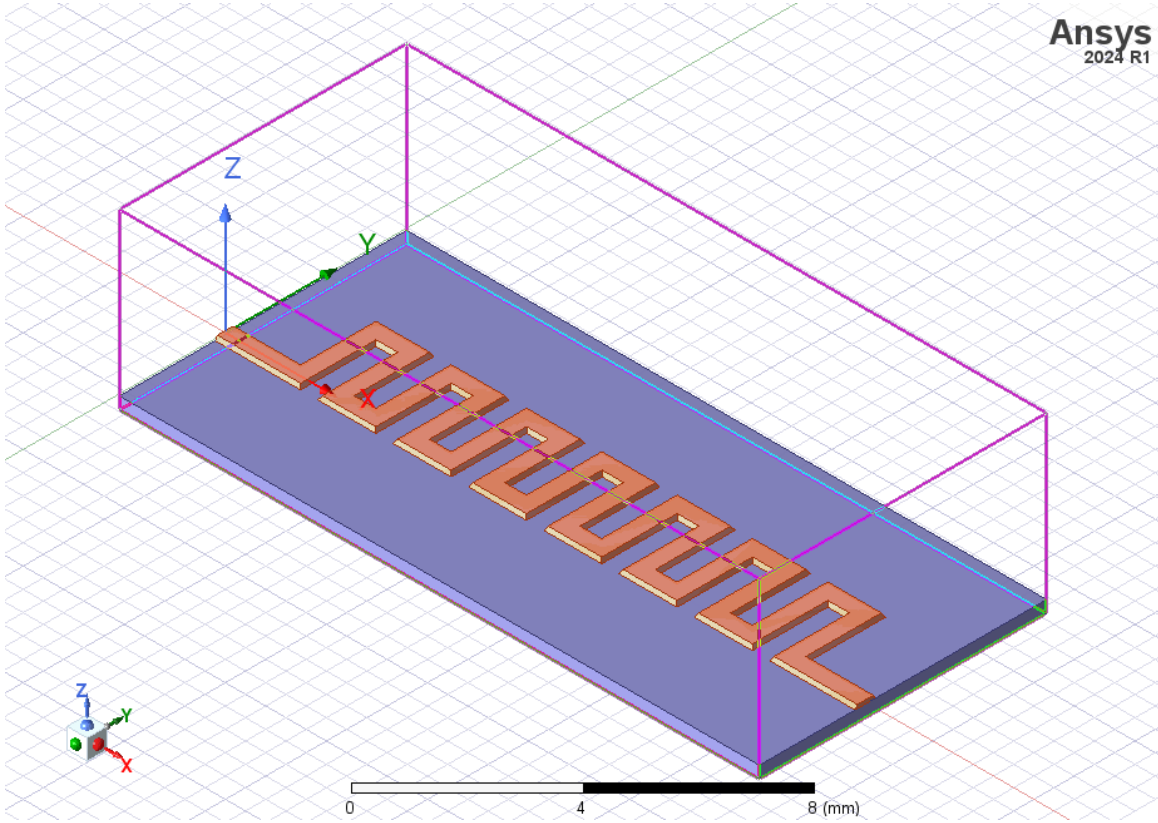
建立 Python 虛擬環境並安裝 PyAEDT

- Python虛擬環境的好處
  - 建立 Python 虛擬環境能夠提供專案隔離的優勢，讓不同專案的套件和依賴不會互相影響。每個虛擬環境都是獨立的，這不僅確保了穩定性，也讓你可以靈活管理不同的 Python 版本和專案需求。虛擬環境的輕量化特性，還能減少系統資源的佔用，專注於專案本身所需的依賴。
- PyAEDT 模組：
  - PyAEDT 是一個專為 ANSYS 電磁場仿真設計的自動化工具，簡化了模型設置、仿真運行和結果處理的過程。無論是在 Windows 還是 Linux 平台上，PyAEDT 都可以輕鬆集成並運行，幫助用戶更高效地進行複雜的電磁模擬分析。強大的 API 支援和詳細的文件資源，讓你能夠快速上手，編寫腳本來自動化各種模擬流程。



參數化模型

- 要訓練數學模型，首先面臨的挑戰是如何對結構進行參數化。
- 在 HFSS 中，對複雜結構進行參數化操作，如果僅依賴內建的編輯功能，往往有其困難。
- 然而，PyAEDT 可以輕鬆解決這個問題。它通過編程的彈性，使得參數化複雜結構變得更加便捷。
- 舉例來說，serpentine line便無法以HFSS內建的編輯操作(延伸、複製、移動、旋轉等)建立出參數化結構。
- PyAEDT可以僅用20行左右的程式碼便生成出結構。



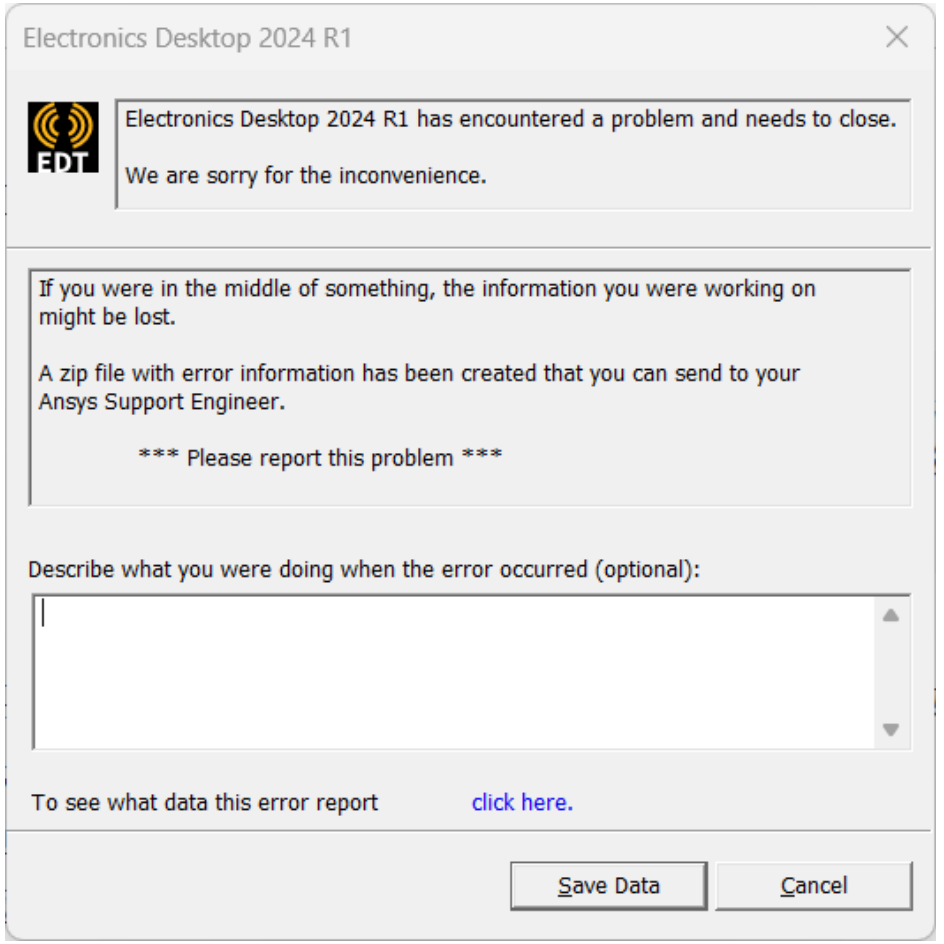
Lab 2. 虛擬環境執行pyaedt程式碼，並檢視HFSS結果

- 將lab2 的範例代碼複製到c:\demo目錄，命名serpentine_1.py
- 開啟command window
- 進到c:\myvenc\Scripts，輸入.\activate啟動虛擬環境
- 輸入python c:\demo\serpentine.py執行
- 檢視HFSS結果

3. 將pyaedt程式碼嵌入optiSLang

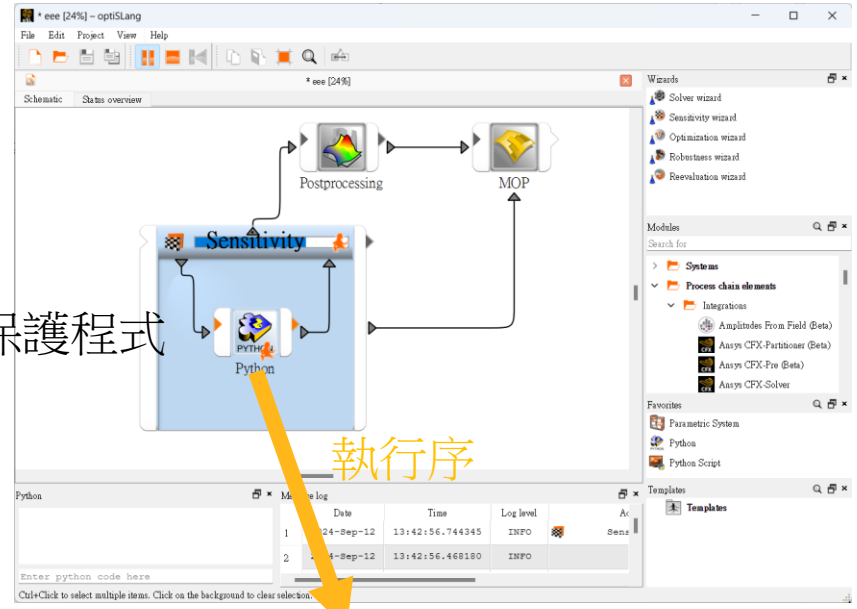
不建議將PyAEDT程式碼直接嵌入Python Node當中

- 我們不建議在 optiSLang 模塊中直接嵌入 PyAEDT 代碼，主要有以下幾個原因：
  - PyAEDT 的執行與虛擬環境的配置密切相關，而 optiSLang 無法直接連結到虛擬環境中的 Python 解釋器。
  - 當參數組合不正確或 AEDT 出現崩潰時，這會導致 optiSLang 的訓練過程中斷，影響訓練流程的連續性與穩定性。
  - optiSLang當中的模塊與專業IDE相比之下過於簡單，不利於複雜代碼管理與檢視。
- 右圖為AEDT崩潰時跳出之警告訊息。



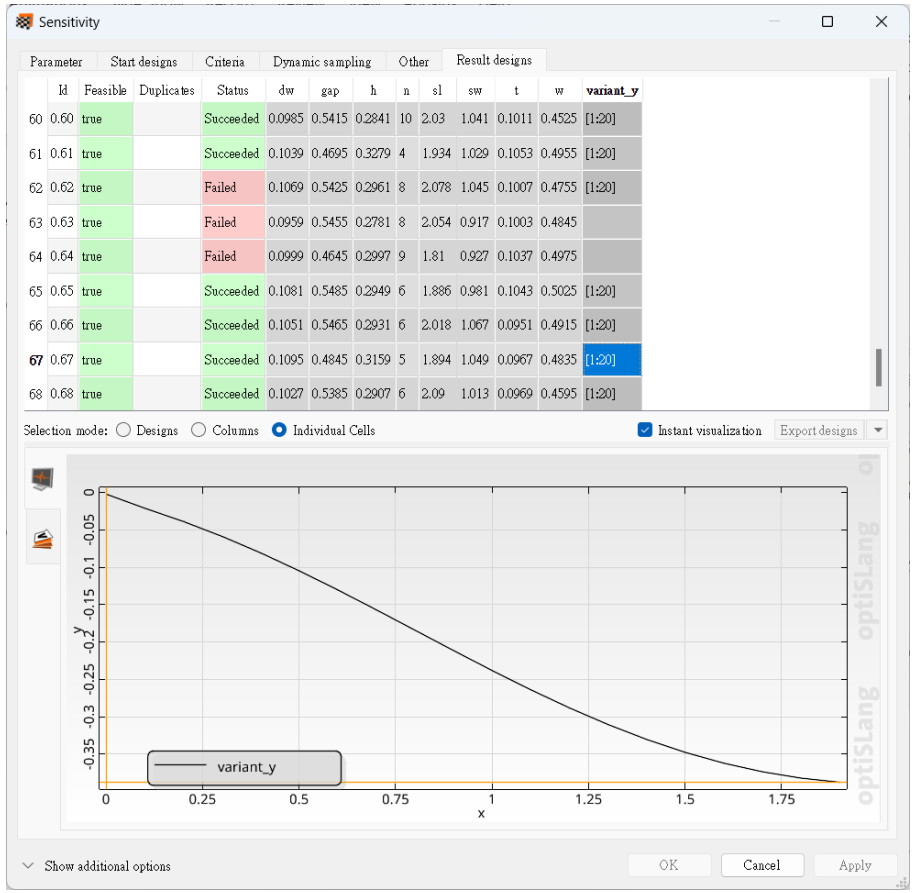
解決方式

- 為了解決 optiSLang 無法直接執行 PyAEDT 代碼的問題，我們可以使用 subprocess 模組來啟動一個獨立的 Python 虛擬環境進行 PyAEDT 相關操作。這樣可以確保 PyAEDT 能夠在正確的環境中執行，而不影響 optiSLang 的內部環境配置。
- 此外，通過捕捉執行過程中的錯誤和超時情況，確保當 AEDT 崩潰或參數錯誤時，我們能夠及時中止進程並自動清理相關資源，如關閉 ansysedt.exe 進程，保持系統穩定運行。



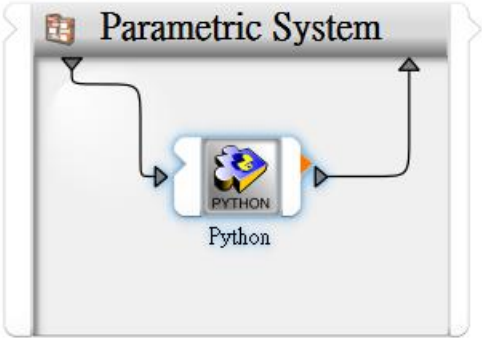
程式碼中的保護機制

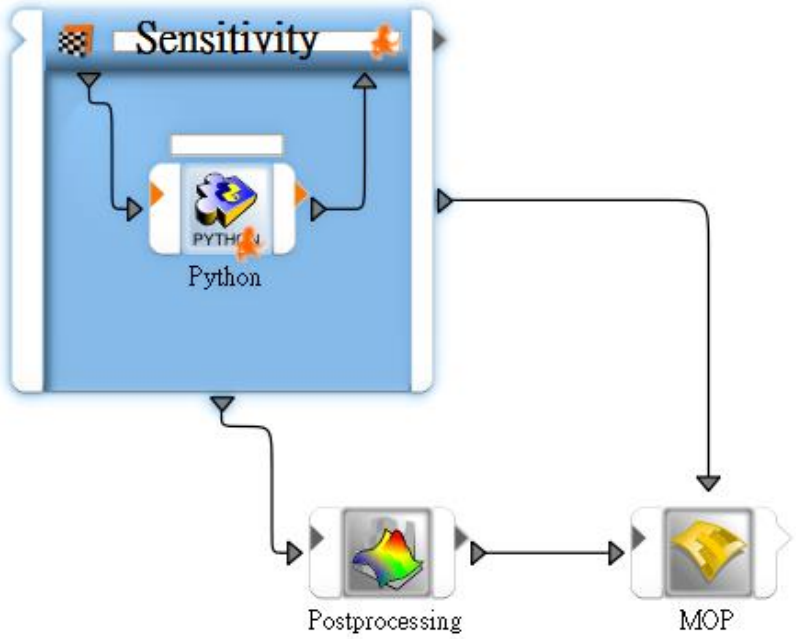
- subprocess.Popen ： 這個方法用來啟動一個外部程序並且在Python中進行互動。在這裡，它被用來執行一個獨立的Python腳本，該腳本以PyAEDT進行模擬運算。這種做法使得模擬運算可以在主程式之外進行，並允許主程式捕捉和處理腳本的標準輸出（stdout）和錯誤信息（stderr）。這樣可以防止主程式因腳本錯誤而崩潰。
- try-except ： 這個部分用來捕捉和處理在執行外部腳本過程中可能發生的錯誤。主要是通過 communicate 方法來等待外部腳本的執行結果，並在遇到問題（如超時、腳本執行失敗）時，通過 except 來捕捉錯誤並執行相應的處理。這樣做可以保證即使腳本執行失敗，主程式也能穩定運行，不會因為未處理的錯誤而中斷。
- timeout 與 proc-kill ： 如果外部腳本在指定時間（如120秒）內未完成，則會觸發 TimeoutExpired 例外，程式會終止這個過程（process），以避免長時間掛起或無限等待。隨後，程式會使用 psutil 模組檢查是否有遺留的 ansysedt.exe 進程，並將其強制終止，這樣可以確保系統資源不被占用並且清理可能導致的AEDT崩潰的殘餘進程，確保後續模擬的正常進行。



Lab 3. 建立Python連接

- 建立Parametric System及Python Node
- 將保護代碼複製到Python Node當中
- 將serpentine_2.py複製到c:/demo目錄當中
- 執行test run
- 設定參數與response
- 在Parametric System設定參數範圍
- 加入Sensitivity並設定1D模型建立相關必要參數
- 執行建模計算

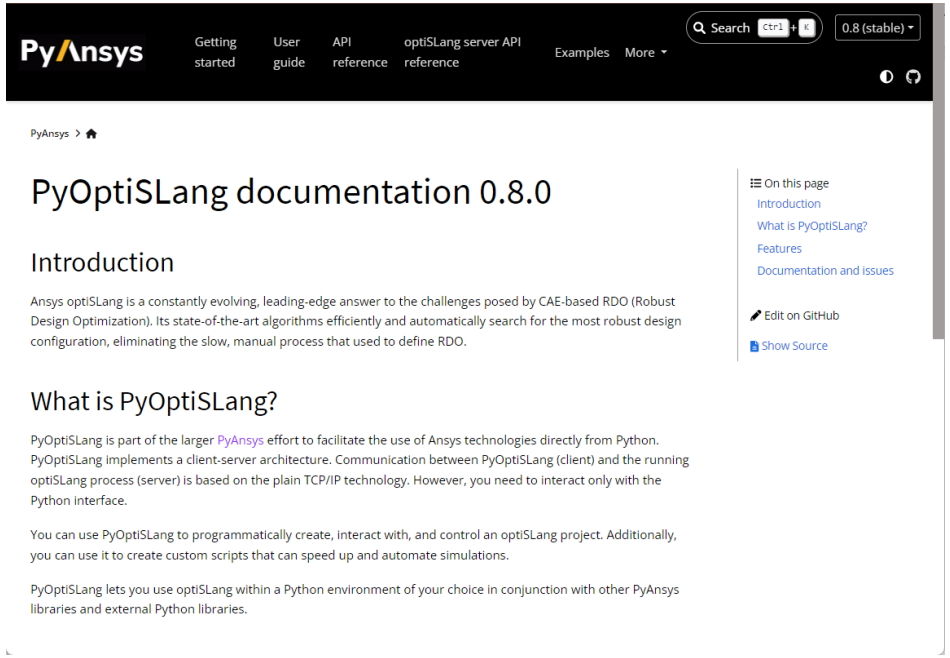




4. PyOptiSLang生成模型與優化

PyOptiSLang介紹

- 為了提升使用 PyAEDT 與 optiSLang 的整合效率，我們可以利用 PyOptiSLang 模組自動化一些過程。在過去，將 Python 代碼手動添加到 optiSLang 的 schematic 中並設置相關參數及其範圍，若參數眾多，這一步驟將會變得相當繁重。但現在，通過使用 PyOptiSLang 模組，我們可以編寫程式碼來自動化這些設置，從而大大減輕手動操作的負擔。這不僅提高了工作效率，還有助於減少因手動設置錯誤而導致的問題。
- PyOptiSLang 採用客戶端-伺服器架構，透過 TCP/IP 技術進行溝通，使用者只需與 Python 介面互動即可。PyOptiSLang 允許使用者以程式化方式創建、互動和控制 optiSLang 專案，也可以用來撰寫自定義腳本，從而加速和自動化模擬流程。此外，它還能與其他 PyAnsys 庫和外部 Python 庫無縫整合，使得 optiSLang 可靈活運行於各種 Python 環境中。



Lab 4a. PyOptiSLang建立MOP模型

- 要在 PyOptiSLang 中建立 MOP（Metamodel of Optimal Prognosis）模型，通常需要執行以下步驟：
  - 引入所需模組：使用 PyOptiSLang 提供的 API 建立與 OptiSLang 的連接，並加載已有的專案或創建新專案。
  - 參數與響應定義：將設計變數和響應註冊為模型中的參數，這些參數將作為 MOP 的輸入與輸出。
  - 構建MOP節點：在模型中創建 AMOP 節點，該節點負責根據設計參數生成 MOP 模型。
  - 設置範圍與目標：為設計參數設置上下界，並添加目標準則，例如最小化或最大化某一響應。
  - 執行MOP訓練：啟動專案並讓 OptiSLang 進行樣本生成與訓練，進而生成 MOP 模型。

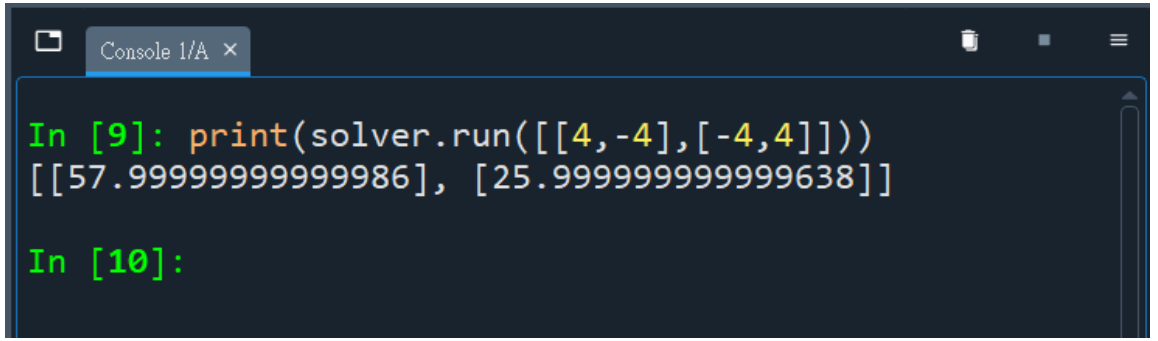
函數說明

- 這段程式主要用於設置和執行 OptiSLang 流程。以下是主要函數的簡短說明：
  - osl_server.new()：創建一個新的 OptiSLang 專案。
  - osl_server.create_node()：創建優化（AMOP）和 Python 節點。
  - osl_server.set_actor_property()：設置節點屬性，例如指定 Python 腳本路徑。
  - osl_server.connect_nodes()：連接節點間的設計資料傳遞路徑。
  - osl_server.register_location_as_parameter()：註冊優化參數（如 x1, x2）。
  - osl_server.register_location_as_response()：註冊響應（如 y）。
  - osl_server.add_criterion()：添加優化準則，設定目標為最小化 y。
  - osl_server.save_as()：保存專案為 .opf 文件。
  - osl.application.project.start()：啟動專案以開始優化流程。
- 修改節點屬性
  - prop = osl_server.get_actor_properties(python)
  - prop['Path']['path']['split_path']['head'] = 'c:/demo'

Lab 4b. 讀取MOP模型

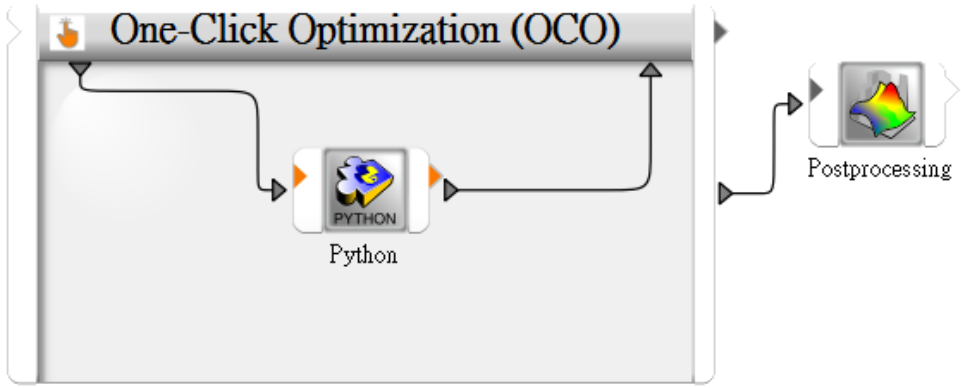
- MOP模型建立後，可以方便地使用 Python 來讀取和操作模型中的值。MOP 是通過數學近似方式，將複雜的數據關係簡化成可以快速運算的模型，因此在參數化優化和多目標優化中應用廣泛。
- 一旦 MOP 模型建構完成，我們可以透過 Python 語言來進行讀取和運算，這不僅提高了分析的靈活性，還能夠在短時間內對大量數據進行評估。

```python

import sys
sys.path.append(r"C:\demo")

from mopsolver import MOPSolver
osl_install_path = r'C:\Program Files\ANSYS Inc\v241\optiSLang'
omdb_file = r"c:\demo\example.opd\AMOP\AMOP.omdb"

solver = MOPSolver(osl_install_path, omdb_file)

print(solver)
print(solver.run([[4,-4],[-4,4]]))

```



Lab 5. PyOptiSLang生成一鍵最佳化專案

- 「一鍵最佳化」(One-Click Optimization, OCO) 是optisLang高度自動化的優化流程，旨在通過幾個關鍵步驟有效地探索和優化設計空間。這個過程從初步取樣開始，分析輸入與輸出參數間的關係，以確定各參數對結果的影響。接著，結合全局搜索和局部搜索策略，全局搜索幫助識別整個參數空間的潛在優化方向，而局部搜索則在這些區域內進一步細化和優化。
- 當現有優化方法表現不佳或已收斂時，OCO會評估並切換到新的優化策略，以提高效率和效果。這個過程會不斷迭代，直到達到預設的最大評估次數。在整個過程中，OCO利用後台的元模型來選擇最佳的代理模型，這些模型評估各參數的重要性，自動選擇最有前途的優化方法和設定。這種方法允許快速、高效地達到設計的最優解。



函數說明

- 範例中的函數用於與 OptiSLang 進行自動化交互：
  - osl_server.open()：開啟 OptiSLang 專案。
  - osl_server.set_actor_property()：設置指定模組的屬性，如腳本路徑。
  - osl_server.register_location_as_parameter()：註冊優化參數。
  - osl_server.register_location_as_response()：註冊響應。
  - osl_server.add_criterion()：設置優化目標（如最小化）。
  - osl_server.save_as()：儲存修改後的專案。
  - osl.application.project.start()：啟動專案進行優化。
- 修改節點屬性
  - info = osl_server.get_actor_properties(oco)
  - container = info['ParameterManager']['parameter_container']
  - container[0]['deterministic_property']['lower_bound'] = -5
  - container[0]['deterministic_property']['upper_bound'] = 5


Ansys