Chapter 12: Integrating SciLab/MATLAB with Python for Scientific Computing

---

Introduction

Python has rapidly emerged as one of the most powerful and versatile programming languages in the field of scientific computing due to its readability, rich ecosystem of libraries (NumPy, SciPy, Matplotlib, Pandas), and ability to integrate with other platforms. SciLab and MATLAB, on the other hand, are specialized tools for numerical computation and simulation. This chapter explores the integration of Python with SciLab and MATLAB, empowering users to leverage the computational strengths of these platforms while maintaining the flexibility of Python scripting.

12.1 Need for Integration of SciLab/MATLAB with Python

- Increasing reliance on **multi-language platforms** in scientific computing.
- Access to **advanced plotting**, **data analysis**, and **machine learning libraries** in Python (e.g., `matplotlib`, `pandas`, `scikit-learn`).
- Interfacing legacy MATLAB or SciLab code with **modern Python-based workflows**.
- Enhancing productivity by using **Python as a controller** and MATLAB/SciLab as computational engines.

12.2 Basics of Python-MATLAB Integration

12.2.1 MATLAB Engine API for Python

- MATLAB provides an **official Python API** that allows Python scripts to start and interact with a MATLAB session.
- Installation:
- ```
  cd "matlabroot/extern/engines/python"
  python setup.py install
  ```
- Basic usage in Python:
- ```
  import matlab.engine
  eng = matlab.engine.start_matlab()
  result = eng.sqrt(16.0)
  print(result)
  ```

12.2.2 Calling MATLAB Functions from Python

- Data types must be **converted** between Python and MATLAB.
- MATLAB engine supports:
  - **Basic numeric types**
  - **Arrays** (`matlab.double`, `matlab.int32`, etc.)
  - **Strings** and **cell arrays**
- Example:
- ```
  a = matlab.double([[1, 2, 3], [4, 5, 6]])
  b = eng.sum(a, 1)
  ```

12.3 Executing MATLAB Scripts in Python

- MATLAB `.m` scripts can be executed using:
- `eng.run('myscript.m', nargout=0)`
- Parameters can be passed via workspace:
- ```
  eng.workspace['x'] = 42
  eng.eval('y = x + 10;', nargout=0)
  result = eng.workspace['y']
  ```

12.4 Integrating SciLab with Python

12.4.1 Installing and Configuring SciLab for Python

- Use the **PyScilab** package or call SciLab via **subprocess interface**.
- SciLab does not offer as seamless an engine API as MATLAB, but integration is still achievable.

12.4.2 Using `subprocess` to Call SciLab from Python

- Run SciLab scripts through command line:
- ```
  import subprocess
  subprocess.run(["scilab-cli", "-f", "myscript.sce"])
  ```
- Data exchange can be handled using **files (CSV, TXT)** or **command-line arguments**.

  ```

12.5 Data Exchange between Python and MATLAB/SciLab

12.5.1 File-Based Communication

- Use `.mat` files to share data:
  - Python: `scipy.io.savemat`, `loadmat`
  - MATLAB: `save`, `load`
- For SciLab, use `.csv` or `.txt` files.

12.5.2 Shared Data via APIs

- MATLAB Engine API supports:
  - Importing data to workspace
  - Extracting output data

12.6 Visualization and Plotting

12.6.1 Using MATLAB Plots in Python

- MATLAB plots can be generated from Python calls:
- ```
  eng.plot(matlab.double([1, 2, 3]), matlab.double([4, 5, 6]),
  nargout=0)
  eng.grid(nargout=0)
  ```

12.6.2 Transferring Results to Python for Visualization

- Data generated in MATLAB/SciLab can be retrieved and plotted using `matplotlib`.

12.7 Use Cases and Applications

12.7.1 Signal Processing Example

- Compute FFT in MATLAB and analyze in Python:
  - Generate signal in MATLAB
  - Transfer result to Python for advanced plotting and comparison

12.7.2 Control Systems

- Run simulations in MATLAB/SciLab
- Use Python to optimize parameters or use machine learning models to tune controllers

---
  ```

12.8 Advantages and Challenges of Integration

12.8.1 Advantages

- Flexibility and power of Python + domain-specific MATLAB/SciLab functions
- Access to large ecosystem
- Reuse of existing code base

12.8.2 Challenges

- **Data type conversion overhead**
- **Performance issues** with large datasets
- **Version compatibility** between MATLAB/Python
- Less native support in SciLab (compared to MATLAB)

12.9 Best Practices

- Use **modular scripts** in MATLAB/SciLab for easy calling
- Validate **data conversion formats** before large-scale usage
- Prefer **file-based or API-based structured exchange**
- Maintain **version documentation** of Python and MATLAB/SciLab installations

---