Homework 03

Implementation of a Scientific Computing toolbox

Advanced Programming - SISSA, UniTS, 2025-2026

Pasquale Claudio Africa, Giuseppe Alessio D'Inverno

Due date: 11 Jan 2026

Objective

This assignment builds on Homework 02, integrating new topics like:

- Python and built-in data types.
- Object-oriented programming in Python.
- Python modules and packages.
- C++ and Python integration using pybind11.
- Use of Python ecosystem for scientific computing (NumPy, SciPy, Matplotlib, seaborn, pandas, ...)

Build your Homework 03 on **module A)** and **one module among B), C), D)** from Homework 02 of your choice. You may reuse and extend your Homework 02 code or start fresh - either approach is acceptable.

Your task is to enhance the C++ scientific computing toolbox with Python bindings and additional functionalities.

Tasks

1. **Python bindings using pybind11**: Create Python bindings for the C++ modules, ensuring Python users can seamlessly access all functionalities.
2. **C++-Python integration and performance analysis**: Demonstrate bidirectional interaction (e.g., C++ callbacks in Python or vice versa). Verify that the bindings provide correct results. Discuss the performance balance between C++ and Python, also in view of possible efficiency gains of the hybrid approach.
3. **Object-oriented Python extensions**: Design Python classes complementing your C++ modules, showcasing OOP principles. Leverage Pythonic features: dynamic typing, decorators, magic methods, context managers.
4. **Data analysis and visualization**: Integrate libraries such as NumPy for data manipulation, SciPy for computations, and/or pandas for data analysis. Visualize results using Python plotting libraries (Matplotlib, seaborn).

Requirements

- **Compatibility**: Ensure Python bindings are compatible with the C++ codebase from Homework 02. Target Python 3.8 or higher. Clearly specify the required Python version in your README.
- **Environment**: Use `CMake` as a build system to import pybind11. Clearly specify the commands needed to compile the C++ library and to run the Python code. Include a `requirements.txt` or analogous file listing all Python dependencies.
- **Documentation**: Update the README with instructions on using the Python interface, including installation of the toolbox and any dependencies.
- **Examples and testing**: Provide Python scripts or notebooks demonstrating the use of the toolbox and its correctness. Include tests verifying that the Python interface correctly interacts with the C++ modules.
- **Third-party libraries**: The integration of third-party C++ libraries or Python packages is highly encouraged.

Submission

1. Include a `README` file that:
    - Clearly states **which module(s)** you implemented.
    - Lists all **group members** and their **individual contribution** to the project.
    - Provides **instructions** to compile and run the toolbox.
    - Provides a concise **discussion of the obtained results**, focusing on design choices and considerations about the performance balance between C++ and Python.
2. You must submit both the C++ code and the Python bindings. Your submission should be self-contained and not reference your Hw02 submission. Submit a **single** compressed file (named `Homework_03_Surname1_Surname2.ext`) containing **all** source code (possibly organizing in proper subfolders the C++ and Python parts), the `README`, and any other relevant files or third-party libraries (please comply to their licences).

Evaluation grid (1/3)

1. **C++ functionality and correctness** (up to **1.5 points**):
    - Successful compilation, modules work correctly with accurate results (1 point).
    - Proper error handling and edge case management in C++ code (0.5 points).
2. **Python bindings with pybind11** (up to **1.5 points**):
    - Python bindings compile and work successfully, can be imported without errors, and all key C++ functionalities are accessible from Python (1 point).
    - Proper handling of data type conversions between C++ and Python (0.5 points).

Evaluation grid (2/3)

3. **Integration and interoperability** (up to **1.0 point**):
    - Working Python scripts or Jupyter notebooks demonstrating correctness of C++-Python integration (0.5 points).
    - Demonstrate bidirectional interaction (e.g., Python callbacks in C++, C++ objects manipulated in Python) with efficient data exchange (0.25 points).
    - Analysis of performance balance between C++ and Python (0.25 points).
4. **Python extensions and ecosystem integration** (up to **0.5 points**):
    - Object-oriented Python design with proper OOP principles and use of *Pythonic* features (magic methods, decorators, context managers, properties) (0.25 points).
    - Integration with scientific Python libraries (NumPy, SciPy, pandas, Matplotlib/seaborn) (0.25 points).

Evaluation grid (3/3)

5. **Code quality, organization, and documentation** (up to **0.5 points**):
    - Clear project structure separating C++ and Python components with minimal but meaningful inline comments and docstrings (0.25 points).
    - Comprehensive README with installation instructions, dependencies, and usage examples (0.25 points).
6. **Bonus features** (up to **0.5 points**):
    - **Performance profiling** (0.15 points): Profile the code using tools to identify possible computational bottlenecks either in the C++ or the Python components.
    - **Automated testing** (0.15 points): Implement automated testing either for the C++ or the Python components.
    - **Package distribution** (0.20 points): Create a `pip`-installable Python package using `setuptools` or `poetry` with proper `setup.py` / `pyproject.toml`.