Homework 03

Implementation of a Scientific Computing Toolbox

Advanced Programming - SISSA, UniTS, 2023-2024

Pasquale Claudio Africa, Marco Feder

Due date: 14 Jan 2024

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.
- Use of Python ecosystem for scientific computing (NumPy, SciPy, Matplotlib, seaborn, pandas, ...)
- C++ and Python integration using pybind11.

Your task is to enhance the C++ scientific computing toolbox developed in Homework 02 with Python bindings and additional functionalities.

Build your Homework 03 on **module A)** and **one module among B), C), D)** from Homework 02, of your choice.

Tasks (1/2)

1. **Python bindings using pybind11**:
    - Create Python bindings for the C++ modules using pybind11.
    - Ensure Python users can seamlessly use the functionalities of these modules.
2. **Advanced Python-C++ integration**:
    - Demonstrate integration where Python and C++ interact more complexly, like C++ callbacks being used in Python or vice versa.
    - Explore efficiency gains from this hybrid approach in data-intensive tasks.

Tasks (2/2)

3. **Object-oriented Python extensions**:
    - Design Python classes that complement your C++ modules, showcasing OOP principles.
    - Implement features that leverage Python's flexibility, like dynamic typing, decorators, magic methods, and context management.
4. **Data analysis and visualization in Python**:
    - Add functionality to visualize results using a Python plotting library (e.g., Matplotlib, seaborn).
    - Integrate NumPy for data manipulation, SciPy for advanced computations, and/or pandas for data analysis.

Requirements

- **Compatibility**: Ensure Python bindings are compatible with the C++ codebase from Homework 02.
- **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 in real-world scenarios. 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 a concise **discussion of the obtained results**, with a focus on design choices and considerations about the performance balance between C++ and Python.
2. Use `CMake` as a build system to detect pybind11 and exploit its functionalities. Clearly specify the commands needed to compile the C++ library and and to run the Python code successfully.
3. 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. **Functionality** (up to **10 points**): C++ code and Python bindings work correctly with all modules.
2. **Integration** (up to **10 points**): Seamless integration between Python and C++ components.
3. **Code organization, code quality and documentation** (up to **5 points**): Clean, well-organized code and clear, instructive documentation.
4. **Examples and testing** (up to **5 points**): Practical examples with adequate tests.
5. **Bonus** points (up to **4 points**):
    - Profile the code to detect computational bottlenecks and optimize the performance.
    - Implement automated testing frameworks to ensure the robustness of the code.
    - Create a Python package of the toolbox using `setuptools`, possibly making the package easily installable via `pip`.