




# Artifact Description for: “PyFMM: A Python module for Frequency-Modulated Möbius Signal Decomposition”

Christian Canedo  (Universidad de Valladolid), Rocío Carratalá-Sáez  (Universitat de València),  
Cristina Rueda  (Universidad de Valladolid)

This document provides an artifact description for the associated paper. It includes a detailed outline of the folder structure, the Python modules provided, and the executable scripts needed to reproduce the results reported in the paper.

The package covers both the computational experiments used to benchmark execution times under different configurations, and the practical examples on real-world data that illustrate the functionality and applicability of the PyFMM software.

## 1 Folder Structure

The repository is organized to provide the necessary data, core modules, and all scripts used to reproduce the computational experiments. The main structure includes the input data files, the PyFMM module, the AFDCal and 3DFMM folders for alternative implementations, the output directory with result files and plots, and a series of scripts that run the ECG and XPS test cases and the computational benchmarking.

```
Software/
├── Data/
│   ├── ECG_data.csv
│   └── XPS_Fe2p_data.csv
├── PyFMM/
│   └── fit_fmm.py
│   ...
├── AFDCal
├── 3DFMM
├── Results/
│   ├── Timing results (csv)
│   └── Figures/
│       └── Paper plots
├── 01_ECG_case.py
├── 02_XPS_case.py
├── 03_Times_PyFMM.py
├── 04_Times_Comparison.py
├── 04B_Times_Comparison.py
└── 05_Times_Linears.py
```

## 2 Before running

Before running any benchmark scripts, it is necessary to set the working directory to the main **Software** folder so that the local modules and data files can be correctly located. In code:

```
import os
os.chdir("Route/to/Software/")
```

The `fit_fmm()` function of the **PyFMM** module can then be imported as:

```
from PyFMM.fit_fmm import fit_fmm
```

Each script includes its own required imports and setup commands, so some lines may appear repeated across different benchmark files.

## 3 Reproducing Results

The example scripts `01_ECG_case.py` and `02_XPS_case.py` include all the code snippets shown in the manuscript, their results, and reproduce the Figures 5 to 9 presented in Section 4 of the main paper.

Running `02_XPS_case.py` is more demanding (it takes around 1–2 minutes), while `01_ECG_case.py` executes almost immediately.

Can be executed as follows:

```
$ python 01_ECG_case.py
$ python 02_XPS_case.py
```

## 4 Running benchmarks

For the computational timing benchmarks, the script `03_Times_PyFMM.py` generates the runtime measurements and plots for Figure 4, which analyzes the scalability of the model without parameter restrictions. The

scripts `04_Times_Comparison.py` and `04B_Times_Comparison.py` run comparisons between the original AFD and FMM implementations (where only selected configurations can be reproduced) and the **PyFMM** version with nonlinear parameter restrictions. Their output consists of plain text files reporting the mean and maximum runtimes for each experiment.

Finally, `05_Times_Linears.py` measures the execution time for models with linear parameter restrictions. Fully reproducing these results can be computationally demanding; it is recommended to reduce **N\_REPEATS** to the minimum if you only wish to check that the scripts run as expected.

All benchmark scripts can be launched directly from the command line. For example:

```
$ python 03_Times_PyFMM.py
$ python 05_Times_Linears.py 5
```

In this example, the optional number **5** sets the **N\_REPEATS** value to 5 instead of the default 100. If no argument is provided, the scripts use **N\_REPEATS = 100** by default.