The Journal of Open Source Software

pyomeca: An Open-Source Framework for Biomechanical Analysis

Romain Martinez[1], **Benjamin Michaud**[1], **and Mickael Begon**[1]

1 School of Kinesiology and Exercise Science, Faculty of Medicine, University of Montreal, Canada

DOI: 10.21105/joss.02431

Software

- Review
- Repository
- Archive

Editor: Kevin M. Moerman

Reviewers:

- @BKillen05
- @mitkof6

Submitted: 09 June 2020
Published: 07 August 2020

Statement of Need

Biomechanics is defined as the study of the structure and function of biological systems by means of the methods of mechanics (Hatze, 1974). While biomechanics branches into several subfields, the data used are remarkably similar. The processing, analysis and visualization of these data could therefore be unified in a software package. Most biomechanical data characterizing human and animal movement appear as temporal waveforms representing specific measures such as muscle activity or joint angles. These data are typically multidimensional arrays structured around labels with arbitrary metadata (Figure 1). Existing software solutions share some limitations. Some of them are not free of charge (Damsgaard, Rasmussen, Christensen, Surma, & Zee, 2006) or based on closed-source programming language (Dixon, Loh, Michaud-Paquette, & Pearsall, 2017; Muller, Pontonnier, Puchaud, & Dumont, 2019). Others do not leverage labels and metadata (Hachaj & Ogiela, 2019; Virtanen et al., 2020; Walt, Colbert, & Varoquaux, 2011). pyomeca is a python package designed to address these limitations.

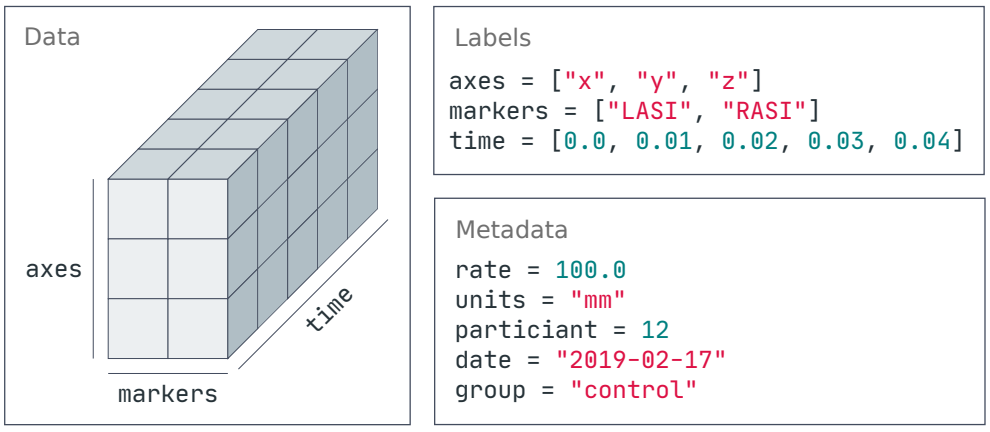

Figure 1: An example of biomechanical data with skin marker positions. These data are inherently multidimensional and structured around labels. Metadata are also needed to inform about important features of the experiment.

Summary

As a python library, pyomeca enables extraction, processing and visualization of biomechanical data for use in research and education. It is motivated by the need for simpler tools and more

reproducible workflows allowing practitioners to focus on their specific interests and leaving pyomeca to handle the computational details for them. pyomeca builds on the core scientific python packages, in particular numpy (Walt et al., 2011), scipy (Virtanen et al., 2020), matplotlib (Hunter, 2007) and xarray (Hoyer & Hamman, 2017). By providing labeled querying and computation, efficient algorithms and persistent metadata, the integration of xarray facilitates usability, which is a step towards the adoption of programming in biomechanics. xarray is designed as a general-purpose library and tries to avoid including domain specific functionalities — but inevitably, the need for more domain specific logic arises. pyomeca provides a biomechanics layer that supports specialized file formats (c3d, mat, trc, sto, mot, csv and xlsx) and implements signal processing and matrix manipulation routines commonly used in biomechanics. pyomeca was written in a modular, object-oriented way, which makes it extensible and promotes the use of method chaining. pyomeca follows software best practices by being fully tested, linted and type annotated — ensuring that the package is easily distributable and modifiable. In addition to the static documentation and API reference, pyomeca includes a set of Jupyter Notebooks with examples. These notebooks can be read and executed by anyone with only a web browser through binder.

Features

pyomeca inherits from the xarray features set, which includes label-based indexing, arithmetic, aggregation and alignment, resampling and rolling window operations, plotting, missing data handling and out-of-core computation. In addition, pyomeca has four data structures built upon xarray. Each structure is associated with a specific biomechanical data type:

- Angles: joint angles,
- Rototrans: rototranslation matrix,
- Analogs: generic signals such as EMGs, force signals or any other analog signals,
- Markers: skin markers position.

While there are technically dozens of functions implemented in pyomeca, one can generally group them into two distinct categories: object creation and data processing.

Object Creation

The starting point for working with pyomeca is to create an object with one of the specific methods associated with the different classes available. pyomeca offers several ways to create these objects: by directly specifying the data, by sampling random data from distributions, by converting other data structures or by reading files (Figure 2).

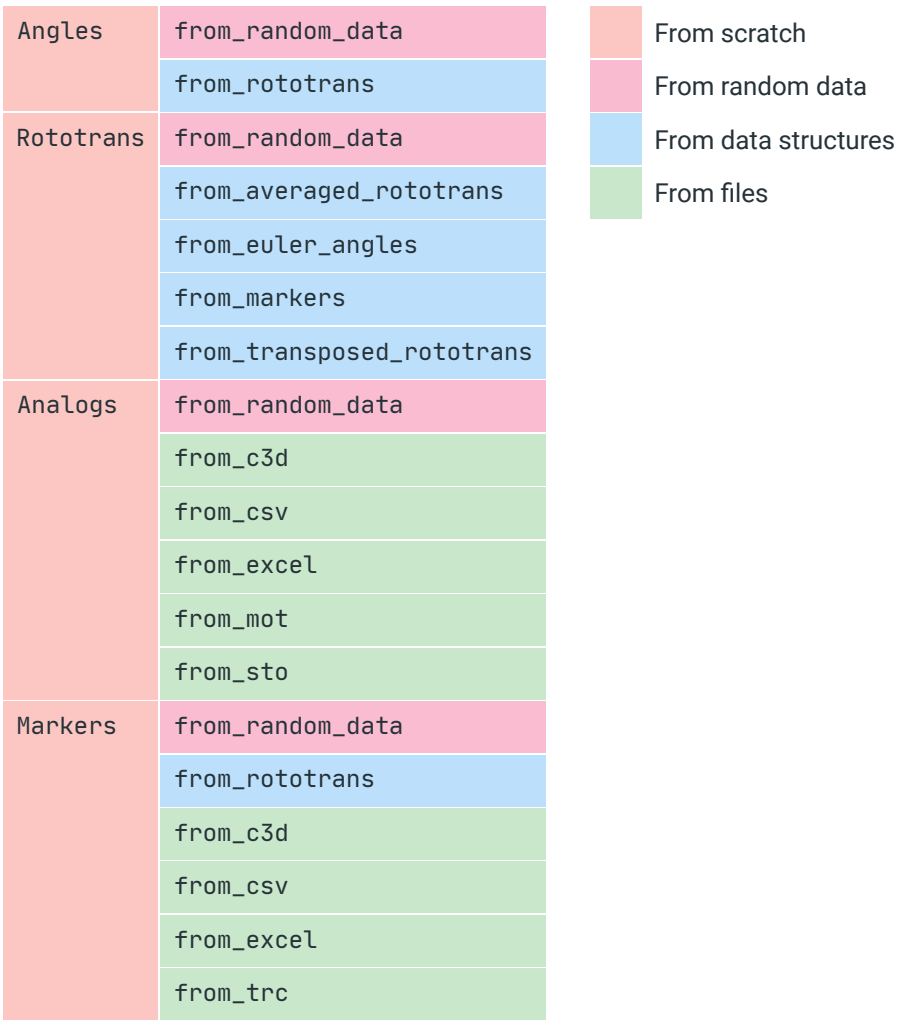

Figure 2: `pyomeca` offers several ways to create specialized data structures: from scratch (orange), from random data (red), from other data structures (blue) or from files (green).

Data Processing

`pyomeca`'s main functionality is to offer dedicated biomechanical routines. These features can be broadly grouped into different categories: filtering, signal processing, normalization, matrix manipulation and file output functions (Figure 3).

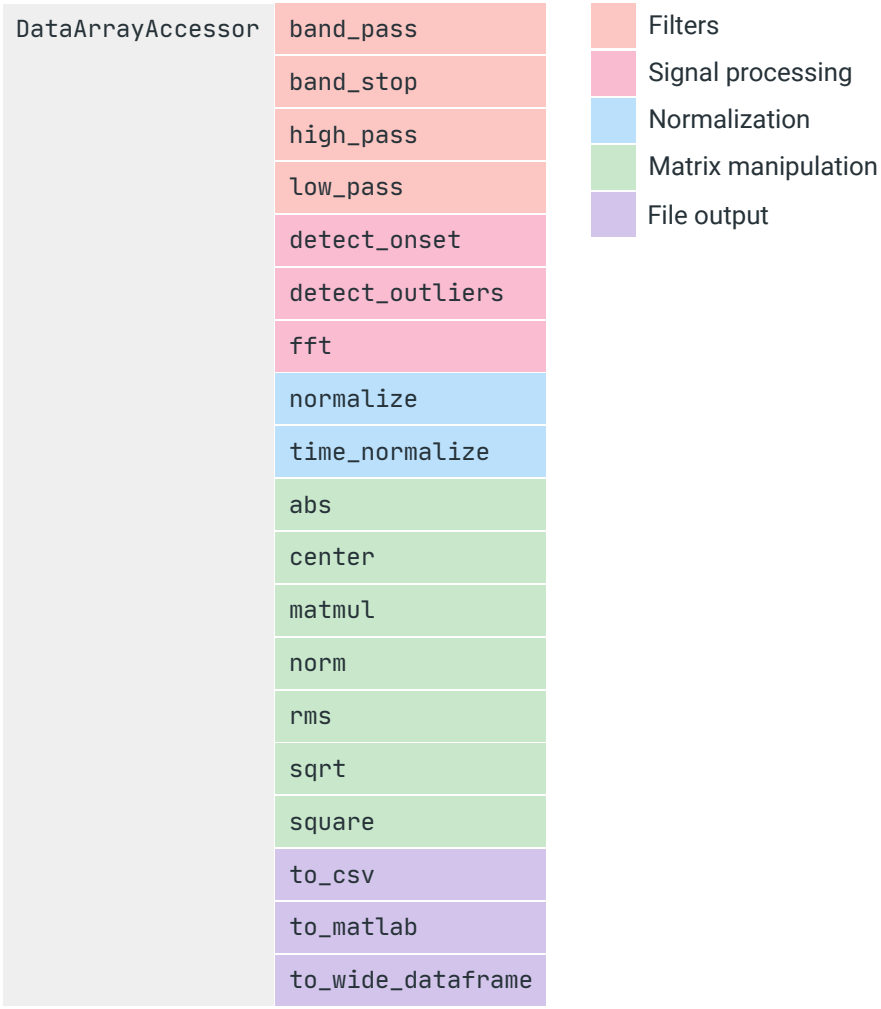

Figure 3: `pyomeca` data processing capabilities are available through the `meca` `DataArrayAccessor` (e.g. `array.meca`) that allow implementing domain specific methods on `xarray` data objects. These methods can be categorized into filters (orange), signal processing (red), normalization (blue), matrix manipulation (green) and file output (purple) routines.

A Biomechanical Example: Electromyographic Pipeline

`pyomeca` has documented examples for different biomechanical tasks such as getting Euler angles from a rototranslation matrix, creating a system of axes from skin markers position or setting a rotation or a translation. Another typical task concerns electromyographic (EMG) data processing. Using `pyomeca`, one can easily extract (Figure 4), process (Figure 5) and visualize (Figure 6, Figure 7 and Figure 8) such data.

```
from pyomeca import Analogs

emg = Analogs.from_c3d("data.c3d")
emg.plot(x="time", hue="channel")
```

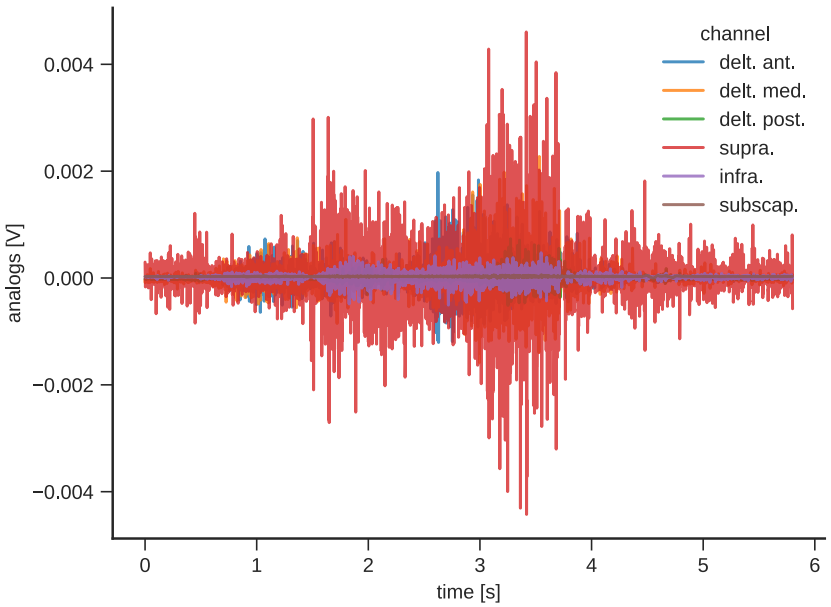

Figure 4: Biomechanical data are often stored in the `c3d` binary file format. Thanks to the `ezc3d` library (Michaud & Begon, 2020), `pyomeca` can easily read these files and visualize them with the `matplotlib` interface provided by `xarray`.

```
emg_processed = (
    emg.meca.band_pass(order=2, cutoff=[10, 425])
    .meca.center()
    .meca.abs()
    .meca.low_pass(order=4, cutoff=5)
    .meca.normalize()
)
emg_processed.plot(x="time", col="channel", col_wrap=3)
```

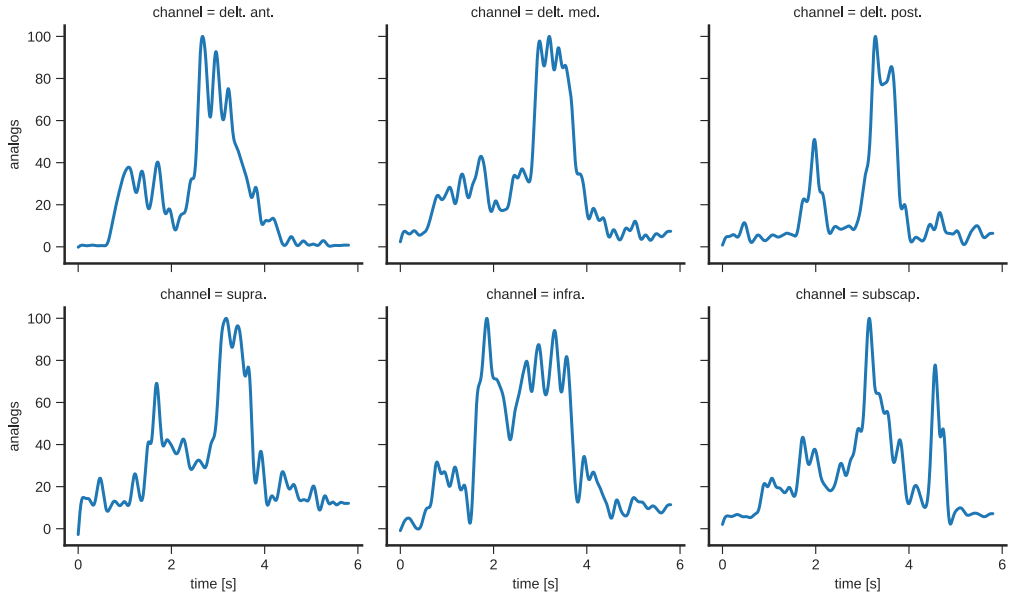

Figure 5: EMG data analysis consists of a series of signal processing steps that can be carried out by pyomeca in a clear and modular way.

```
import matplotlib.pyplot as plt

_, axes = plt.subplots(ncols=2)

emg_processed.mean("channel").plot(ax=axes[0])
emg_processed.plot.hist(ax=axes[1], bins=50)
```

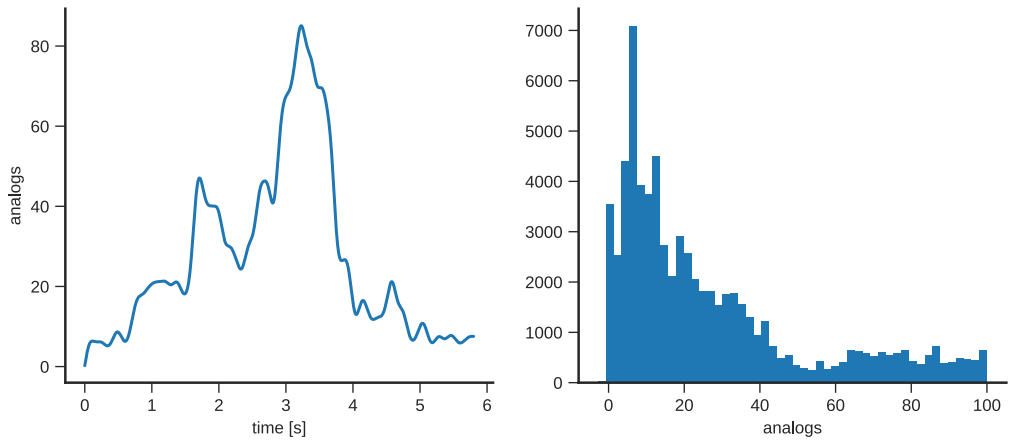

Figure 6: It is straightforward to represent the average profile of the EMG signal (left) or the distribution of EMG activations (right) thanks to xarray.

```
emg_dataframe = emg_processed.meca.to_wide_dataframe()
emg_dataframe.plot.box(showfliers=False)
```

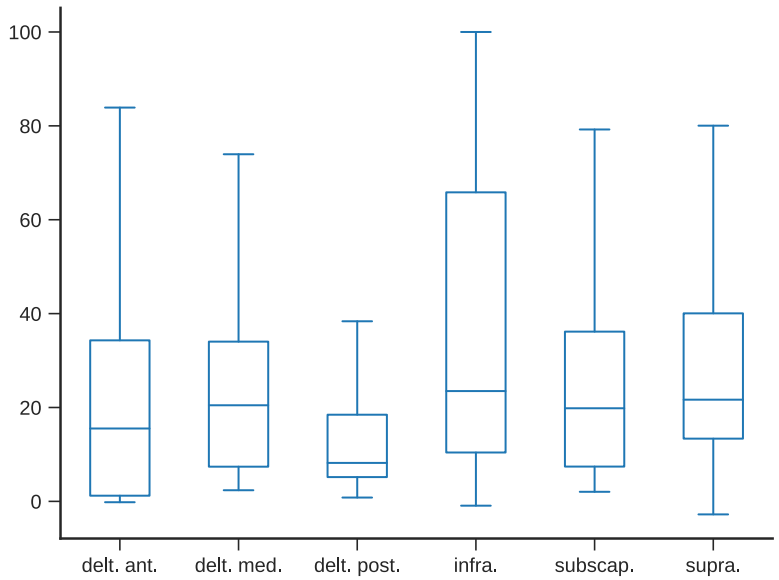

Figure 7: `pyomeca` offers a method to convert the data structure into a `pandas` dataframe (McKinney, 2010). This allows users to further extend the plot possibilities using the visualization built into `pandas` itself, such as boxplot.

```
emg_dataframe.corr().style.background_gradient().set_precision(2)
```

	delt. ant.	delt. med.	delt. post.	infra.	subscap.	supra.
delt. ant.	1.0	0.78	0.38	0.74	0.6	0.6
delt. med.	0.78	1.0	0.77	0.74	0.76	0.9
delt. post.	0.38	0.77	1.0	0.62	0.67	0.84
infra.	0.74	0.74	0.62	1.0	0.61	0.75
subscap.	0.6	0.76	0.67	0.61	1.0	0.78
supra.	0.6	0.9	0.84	0.75	0.78	1.0

Figure 8: By using a `pandas` dataframe, users also benefit from its broad range of IO tools and statistical methods, such as computing the correlation matrix between the different muscles.

Research Projects Using `pyomeca`

You can find an up-to-date list of research projects using `pyomeca` on the static documentation.

Acknowledgements

`pyomeca` is an open-source project created and supported by the Simulation and Movement Modeling (S2M) lab located in Montreal. We thank the contributors that helped build `pyomeca`. You can find an up-to-date list of contributors on GitHub. We also would like to extend thanks to the contributors of the libraries used to build `pyomeca` — particularly `numpy`, `scipy`, `matplotlib` and `xarray`.

References

Damsgaard, M., Rasmussen, J., Christensen, S. T., Surma, E., & Zee, M. de. (2006). Analysis of musculoskeletal systems in the AnyBody modeling system. *Simulation Modelling Practice and Theory*, *14*(8), 1100–1111. doi:10.1016/j.simpat.2006.09.001

Dixon, P. C., Loh, J. J., Michaud-Paquette, Y., & Pearsall, D. J. (2017). BiomechZoo: An open-source toolbox for the processing, analysis, and visualization of biomechanical movement data. *Comput. Methods Programs Biomed.*, *140*, 1–10. doi:10.1016/j.cmpb.2016.11.007

Hachaj, T., & Ogiela, M. R. (2019). RMoCap: An R language package for processing and kinematic analyzing motion capture data. *Multimedia Systems*. doi:10.1007/s00530-019-00633-9

Hatze, H. (1974). Letter: The meaning of the term "biomechanics". *J. Biomech.*, *7*(2), 189–190. doi:10.1016/0021-9290(74)90060-8

Hoyer, S., & Hamman, J. J. (2017). Xarray: N-D labeled arrays and datasets in python. *Journal of Open Research Software*, *5*, 304. doi:10.5334/jors.148

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science Engineering*, *9*(3), 90–95. doi:10.1109/MCSE.2007.55

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Proceedings of the python in science conference (pp. 56–61). Austin, Texas: SciPy. doi:10.25080/Majora-92bf1922-00a

Michaud, B., & Begon, M. (2020). EZC3D: Easy to use c3d reader/writer in c++, python and matlab. *GitHub repository*. GitHub. Retrieved from https://github.com/pyomeca/ezc3d

Muller, A., Pontonnier, C., Puchaud, P., & Dumont, G. (2019). CusToM: A matlab toolbox for musculoskeletal simulation. *JOSS*, *4*(33), 927. doi:10.21105/joss.00927

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nat. Methods*, *17*(3), 261–272. doi:10.1038/s41592-019-0686-2

Walt, S. van der, Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. *Comput. Sci. Eng.*, *13*(2), 22–30. doi:10.1109/MCSE.2011.37