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 musculoskeletal 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 proprietary (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). pyomca is a Python package designed to address these limitations. It provides basic operations useful in the daily workflow of a biomechanical researcher such as reading, writing, filtering and plotting, but also more advanced biomechanical routines geared towards rigid body mechanics and signal processing. By offering a single, efficient and flexible implementation, pyomca standardizes these procedures, freeing up valuable research time, thereby allowing researchers to focus on the scientific research questions at hand.

Figure 1: An example of musculoskeletal 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 seems to be relevant in a context where scientists do not always receive extensive training in programming. 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 musculoskeletal biomechanics. We believe pyomeca is a solid foundation for more advanced research implementations and could, in the future, provide functions for processing and extracting meaningful clinical outputs from the raw data. pyomeca was written with the Unix philosophy in mind, that is to design each program to do one thing well and expect the output of every program to become the input to another, as yet unknown, program. As such, it is written in a modular, object-oriented way — which makes it extensible and easy to develop — and it interacts well with other projects under the pyomeca umbrella, namely ezc3d, pyosim or biorbd-viz. 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 feature 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 electromyographic (EMG) and force signals or any other analog signals,
- **Markers**: skin markers positions.

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).

---

### 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 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).
<table>
<thead>
<tr>
<th>DataArrayAccessor</th>
<th>band_pass</th>
<th>band_stop</th>
<th>high_pass</th>
<th>low_pass</th>
<th>detect_onset</th>
<th>detect_outliers</th>
<th>fft</th>
<th>normalize</th>
<th>time_normalize</th>
<th>abs</th>
<th>center</th>
<th>matmul</th>
<th>norm</th>
<th>rms</th>
<th>sqrt</th>
<th>square</th>
<th>to_csv</th>
<th>to_matlab</th>
<th>to_wide_dataframe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filters</td>
<td>Signal processing</td>
<td>Normalization</td>
<td>Matrix manipulation</td>
<td>File output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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 positions or setting a rotation or a translation. Another typical task concerns 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.

```python
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.

\[
\text{emg\_processed} = (\text{emg\_meca}\text{.band\_pass(order=2, cutoff=[10, 425])}) \text{.meca}\.center() \text{.meca.abs()} \text{.meca}\text{.low\_pass(order=4, cutoff=5)} \text{.meca}\text{.normalize()})
\]

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.

```python
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.

```python
emg_dataframe = emg_processed.mecca.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)

<table>
<thead>
<tr>
<th></th>
<th>delt. ant</th>
<th>delt. med</th>
<th>delt. post</th>
<th>infra</th>
<th>subscap</th>
<th>supra</th>
</tr>
</thead>
<tbody>
<tr>
<td>delt. ant</td>
<td>1.0</td>
<td>0.78</td>
<td>0.38</td>
<td>0.74</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>delt. med</td>
<td>0.78</td>
<td>1.0</td>
<td>0.77</td>
<td>0.74</td>
<td>0.76</td>
<td>0.9</td>
</tr>
<tr>
<td>delt. post</td>
<td>0.38</td>
<td>0.77</td>
<td>1.0</td>
<td>0.62</td>
<td>0.67</td>
<td>0.84</td>
</tr>
<tr>
<td>infra</td>
<td>0.74</td>
<td>0.74</td>
<td>0.62</td>
<td>1.0</td>
<td>0.61</td>
<td>0.75</td>
</tr>
<tr>
<td>subscap</td>
<td>0.6</td>
<td>0.76</td>
<td>0.67</td>
<td>0.61</td>
<td>1.0</td>
<td>0.78</td>
</tr>
<tr>
<td>supra</td>
<td>0.6</td>
<td>0.9</td>
<td>0.84</td>
<td>0.75</td>
<td>0.78</td>
<td>1.0</td>
</tr>
</tbody>
</table>

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 (Walt et al., 2011), scipy (Virtanen et al., 2020), matplotlib (Hunter, 2007) and xarray (Hoyer & Hamman, 2017).

https://doi.org/10.21105/joss.02431
References


