PYCV: a PLUMED 2 Module Enabling the Rapid Prototyping of Collective Variables in Python

Toni Giorgino[1]

1 Institute of Biophysics (IBF-CNR), National Research Council of Italy

DOI: 10.21105/joss.01773

Software

- Review
- Repository
- Archive

Submitted: 21 August 2019
Published: 11 October 2019

Summary

Collective variables (CVs) are functions of the coordinates of particles in a molecular system. The choice of CV is crucial to capture relevant degrees of freedom of the model being simulated (Barducci, Bonomi, & Parrinello, 2011). This is especially important when employing *biased sampling* techniques such as umbrella sampling or metadynamics (Laio & Parrinello, 2002; Torrie & Valleau, 1977), which apply generalized forces to CVs to enhance the sampling of events otherwise not observable by direct simulation. CVs may be simple geometrical observables (distances, angles, torsions, etc.), but often they are more complex functions designed to capture structural determinants, such as tertiary and quaternary structure of proteins, experimental observables, crystal symmetries, etc. (Bonomi & Camilloni, 2017; Branduardi, Gervasio, & Parrinello, 2007; Pipolo et al., 2017).

Iterative development of CVs therefore accounts for most of the efforts associated with the exploration of molecular systems, and software packages implementing high-level directives to express biasing potentials and CVs markedly simplify the task. In this regard, the PLUMED library (Tribello, Bonomi, Branduardi, Camilloni, & Bussi, 2014) is especially relevant because it provides numerous pre-defined functions, a *lingua franca* to express CV combinations, atom groups and biasing schemes, and an active community (The PLUMED consortium, 2019). Users willing to explore CVs beyond the pre-defined ones have to implement them in C++, together with the corresponding (often cumbersome) derivatives (Giorgino, 2018). Compiled code is however unwieldy for iterative analysis, because it is relatively low-level, error-prone, and inconvenient in exploratory stages.

This paper introduces **PYCV**, a module for the PLUMED 2 library which enables users to define CVs and arbitrary functions in the Python language. CV implementations may thus be modified and tested independently of the main code, with essentially no "test latency". Of note, coordinates are processed as `numpy` arrays, making it convenient to leverage the vast set of linear algebra and numerical algorithms provided by `numpy`, `scipy`, and many other open-source modules. Furthermore, just-in-time compilation and reverse-mode automatic differentiation are easily accessible using Google's JAX library.

Usage

The **PYCV** module registers itself with PLUMED to provide the following actions:

- `PYTHONCV`, to implement single- and multi-component CVs;
- `PYTHONFUNCTION`, to implement arbitrary functions.

The actions are documented in the respective inline manuals (e.g., `plumed manual --action PYTHONCV`). In both cases, an interpreter is first started; the Python module indicated in the `IMPORT=` keyword is then loaded; from it, an user-chosen function (`FUNCTION=`) is called to perform the computations at each timestep. Modules can contain multiple functions and one-time initialization.

Example

A self-explanatory example is provided for illustration below. It is equivalent to the *radius of curvature* example shown in (Giorgino, 2018). Further examples are available in the manual and in `regtest/pycv`.

Plumed input script

The actions are declared in the PLUMED input file (say, `plumed.dat`). Here, one declares a CV labelled `rc`, to be computed by the Python function `curvature.r()`. It will receive a 3-by-3 array with the coordinates of atoms 1, 4 and 3 (orderly, as rows). The CV value will be `PRINT`ed, and the atoms subject to a constant generalized force pushing to increase the curvature.

```
# Start plumed.dat -----------------------------------------
rc: PYTHONCV ATOMS=1,4,3 IMPORT=curvature FUNCTION=r
    PRINT ARG=rc FILE=colvar.out
    RESTRAINT ARG=rc AT=0 KAPPA=0 SLOPE=1
# End plumed.dat -------------------------------------------
```

Function definition

The actual function `r` is defined in the `curvature.py` file. It computes the radius of the circle passing through three given atom coordinates (the three rows of the input argument, with 0-based indexing). Note how matrix operations enable a readable translation of the sine formula.

The function is expected to return two objects, i.e. the value of the CV at the given coordinates (a scalar), and its gradient with respect to each of the 9 coordinates (a 3-by-3 array); here the gradient function is obtained automatically. Although not directly comparable, the equivalent C++ implementation required approximately 160 lines of non-trivial code.

```
# Start curvature.py ----------------------------------------
# Same calculation (and results) as doi:10.1016/j.cpc.2018.02.017

# Import the JAX library
import jax.numpy as np
from jax import grad, jit

# Implementation of the angle function. @jit really improves speed
@jit
def r_f(x):
    r21 = x[0,:]-x[1,:]
    r23 = x[2,:]-x[1,:]
    r13 = x[2,:]-x[0,:]
    cos2theta = np.dot(r21,r23)**2 / (np.dot(r21,r21) * np.dot(r23,r23))
    sin2theta = 1-cos2theta

    R2= np.dot(r13,r13)/sin2theta/4.0
    return np.sqrt(R2)

# Use JAX to auto-gradient it
r_g = grad(r_f)

# PLUMED will call the following function
def r(x):
    return r_f(x), r_g(x)

# End curvature.py ---------------------------------------------
```

Conclusions

PYCV enables Python-based prototyping of CVs in PLUMED 2. This programming model may be an advantage over standard C++-based development in that

1. functions may be prototyped in high-level language, using extensive mathematical libraries, without boilerplate;
2. just-in-time compilation occurs transparently: code changes incur in no compilation and link delays; and
3. CV code may be automatically differentiated in common cases.

Acknowledgements

I would like to thank PLUMED's lead authors (Prof. Giovanni Bussi, Prof. Carlo Camilloni, Prof. Gareth Tribello and Prof. Massimiliano Bonomi) for inputs and discussions.

References

Barducci, A., Bonomi, M., & Parrinello, M. (2011). Metadynamics. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, *1*(5), 826–843. doi:10.1002/wcms.31

Bonomi, M., & Camilloni, C. (2017). Integrative structural and dynamical biology with PLUMED-ISDB. *Bioinformatics*, *33*(24), 3999–4000. doi:10.1093/bioinformatics/btx529

Branduardi, D., Gervasio, F. L., & Parrinello, M. (2007). From A to B in free energy space. *The Journal of Chemical Physics*, *126*(5), 054103. doi:10.1063/1.2432340

Giorgino, T. (2018). How to differentiate collective variables in free energy codes: Computer-algebra code generation and automatic differentiation. *Computer Physics Communications*, *228*, 258–263. doi:10.1016/j.cpc.2018.02.017

Laio, A., & Parrinello, M. (2002). Escaping free-energy minima. *Proceedings of the National Academy of Sciences of the United States of America*, *99*(20), 12562–12566. doi:10.1073/pnas.202427399

Pipolo, S., Salanne, M., Ferlat, G., Klotz, S., Saitta, A., & Pietrucci, F. (2017). Navigating at Will on the Water Phase Diagram. *Physical Review Letters*, *119*(24), 245701. doi:10.1103/PhysRevLett.119.245701

The PLUMED consortium. (2019). Promoting transparency and reproducibility in enhanced molecular simulations. *Nature Methods*, *16*(8), 670. doi:10.1038/s41592-019-0506-8

Torrie, G. M., & Valleau, J. P. (1977). Nonphysical sampling distributions in Monte Carlo free-energy estimation: Umbrella sampling. *Journal of Computational Physics*, *23*(2), 187–199. doi:10.1016/0021-9991(77)90121-8

Tribello, G. A., Bonomi, M., Branduardi, D., Camilloni, C., & Bussi, G. (2014). PLUMED 2: New feathers for an old bird. *Computer Physics Communications*, *185*(2), 604–613. doi:10.1016/j.cpc.2013.09.018