Computer Physics Communications 228 (2018) 258–263

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

How to differentiate collective variables in free energy codes: Computer-algebra code generation and automatic differentiation[☆]

Toni Giorgino [1]

Institute of Neurosciences, National Research Council (CNR-IN), Corso Stati Uniti 4, I-35127, Padua, Italy

ARTICLE INFO

Article history:
Received 2 September 2017
Received in revised form 12 February 2018
Accepted 18 February 2018
Available online 26 February 2018

Keywords:
Molecular dynamics
Free energy
Biased sampling
Metadynamics
Symbolic
C++

ABSTRACT

The proper choice of collective variables (CVs) is central to biased-sampling free energy reconstruction methods in molecular dynamics simulations. The PLUMED 2 library, for instance, provides several sophisticated CV choices, implemented in a C++ framework; however, developing new CVs is still time consuming due to the need to provide code for the analytical derivatives of all functions with respect to atomic coordinates. We present two solutions to this problem, namely (a) symbolic differentiation and code generation, and (b) automatic code differentiation, in both cases leveraging open-source libraries (SymPy and Stan Math, respectively). The two approaches are demonstrated and discussed in detail implementing a realistic example CV, the local radius of curvature of a polymer. Users may use the code as a template to streamline the implementation of their own CVs using high-level constructs and automatic gradient computation.

Program summary
Program Title: Practical approaches to the differentiation of collective variables in free energy codes: computer-algebra code generation and automatic differentiation
Program Files doi: http://dx.doi.org/10.17632/r4r67bvkdn.1
Licensing provisions: GNU Lesser General Public License Version 3 (LGPL-3)
Programming languages: C++, Python
Nature of problem: The C++ implementation of collective variables (CVs, functions of atomic coordinates to be used in biased sampling applications) in biasing libraries for atomistic simulations, such as PLUMED [1], requires computation of both the variable and its gradient with respect to the atomic coordinates; coding and testing the analytical derivatives complicate the implementation of new CVs.
Solution method: The paper shows two approaches to automate the computation of CV gradients, namely, symbolic differentiation with code generation and automatic code differentiation, demonstrating their implementation entirely with open-source software (respectively, SymPy and the Stan Math Library).
Additional comments: The paper's accompanying code serves as an example and template for the methods described in the paper; it is distributed as the two modules `curvature_codegen` and `curvature_autodiff` integrated in PLUMED 2's source tree; the latest version is available at https://github.com/tonigi/plumed2-automatic-gradients .
[1] Tribello GA, Bonomi M, Branduardi D, Camilloni C, Bussi G. PLUMED 2: New feathers for an old bird. Computer Physics Communications. 2014 Feb;185(2):604-13.

1. Introduction

Biased approaches to molecular dynamics (MD) enable the sampling of events whose occurrence would otherwise be prohibitively rare on the time scales affordable by direct (unbiased) integration of the equations of motion. Central to the possibility to obtain converged estimates of thermodynamic quantities is the search of appropriate projections of the system state [1]; in turn, this enables the search of a reaction coordinate to effectively push the specific system out of free energy minima. When an appropriate reaction coordinate is selected, the sampling of a system can be accelerated

[☆] This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

E-mail address: toni.giorgino@cnr.it.

[1] Present address: Istituto di Biofisica, Consiglio Nazionale delle Ricerche (CNR-IBF), c/o Dipartimento di Bioscienze, Università degli Studi di Milano, via Celoria 26, I-20133 Milan, Italy.

https://doi.org/10.1016/j.cpc.2018.02.017
0010-4655/© 2018 Published by Elsevier B.V.

through a number of *biased* sampling methods, such as umbrella sampling [2], metadynamics [3], SuMD [4] and others [5], most of which enable the reconstruction of the free energy landscape, and in some cases the kinetics [6,7], on the space spanned by the chosen variables. The availability of a wide range of functions of atomic coordinates (collective variables or CVs) is thus a valuable asset in the construction of proper reaction coordinates [8].

PLUMED 2 is a widely-used engine to perform biased sampling simulations in atomistic simulations [9]. Part of PLUMED's success is due to the number and variety of collective variables implemented (see e.g. [10–12]), enabling projections of the system state on "axes" of intuitive value, and the number of CVs implemented in PLUMED 2 has been growing steadily (Fig. 1). Users can incorporate their own CVs coding them in C++; however, the implementation of complex functions is complicated by the need to compute gradients with respect to atomic coordinates, which increases the complexity and debugging time of the corresponding source codes.

Here, we present two approaches to automatically implement CV gradients:

1. a *symbolic differentiation with code generation* approach, where the SymPy computer algebra system (CAS) [13] is used to derive the expressions and automatically produce an equivalent C function (Section 2); and
2. an *automatic code differentiation* approach, using the reverse-mode code differentiation capabilities provided by the Stan Math library [14] (Section 3).

We demonstrate the two approaches on the simple (yet non trivial) case of a CV computing the local curvature of a polymer, approximated as the radius of curvature of a circle passing through three consecutive atoms. The two approaches provide identical numerical results and are based on mature and well-known open source libraries. Their different characteristics will be presented in the discussion section.

Example code is made available as open-source, respectively, in PLUMED 2's `curvature_codegen` and `curvature_autodiff` modules; from there, the corresponding source files can be used as templates for the implementation of customized CVs.

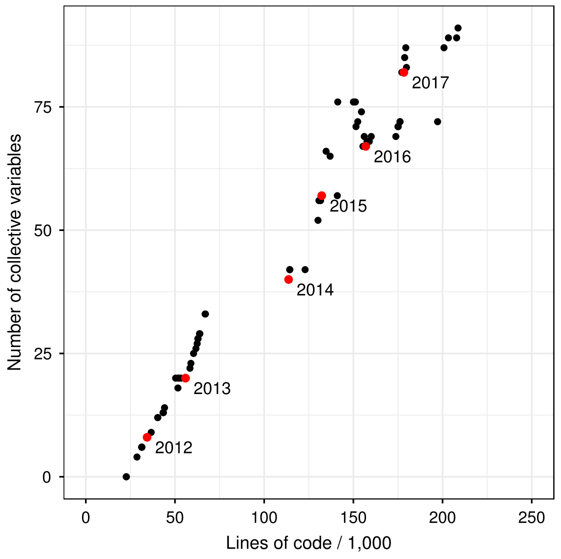

Fig. 1. Growth of the number of CVs in PLUMED 2 and the corresponding code base (lines of C++ code in the master branch, including headers and inline documentation; the count also includes support functions, command line utilities and file readers).

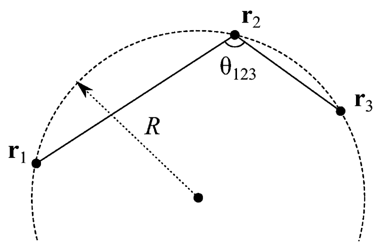

Fig. 2. The radius of curvature as a collective variable $R(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)$.

1.1. Background

A CV is a function of a system's state through the coordinates of its n particles, namely: [15]

$$f(\mathbf{x}) = f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$$

Applying biases to CVs implies that the system is subject to a potential V which depends on the coordinates solely through f:

$$V(\mathbf{x}) = V_1(f(\mathbf{x}))$$

The bias potential translates to forces acting on each atom, which are computed in the biasing library and passed to the molecular dynamics (MD) engine. The MD code adds them to those due to the force field, and integrates the equations of motion. From the chain differentiation rule,

$$\mathbf{F}(\mathbf{x}) = -\nabla_{\mathbf{x}} V_1(f(\mathbf{x})) = -\frac{\partial V_1(f)}{\partial f} \nabla_{\mathbf{x}} f(\mathbf{x})$$

here $\partial V_1/\partial f$ is the *generalized force*, set by the chosen biasing scheme (e.g., a time-dependent sum of Gaussians in the case of metadynamics), while $\nabla_{\mathbf{x}} f$ depends only on the functional form of f and the system state $\mathbf{x}$. Implementation of a CV requires the programmer to write code for $f(\mathbf{x})$ and its derivatives with respect to all of the arguments (number of involved atoms times three Cartesian components).

1.2. Radius of curvature

To illustrate the methods, we shall use as an example the radius of curvature at a given atom along a polymer. A natural choice for this quantity is to compute the radius R of the circle (circumcircle) passing through three given points $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$ (Fig. 2), e.g. the centers of consecutive Cα atoms. The diameter $2R$ is obtained elementarily via the sine rule as the ratio between a side of the triangle formed by the points and the sine of the opposing angle, i.e., calling $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ and θ_{123} the angle at $\mathbf{r}_2$,

$$2R = \frac{|\mathbf{r}_{13}|}{\sin\theta_{123}} \qquad \text{with} \qquad \cos\theta_{123} = \frac{\mathbf{r}_{12} \cdot \mathbf{r}_{23}}{|\mathbf{r}_{12}||\mathbf{r}_{23}|} \tag{1}$$

The above expression is compact in vector notation, but the expressions for its gradient in Cartesian coordinates, i.e. the components of $\nabla_{\mathbf{x}} R(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)$ with $\mathbf{x} = (r_{1x}, r_{1y}, \ldots, r_{3z})$, are unwieldy (shown in full in the notebook `CurvatureCodegen.ipynb`).

1.3. Edge cases and inverse radius

Computer-assisted code generation does not automatically guarantee that the functions are well-defined in all conditions. Of special relevance are singularities on edge cases, such as collinearity ($R \to \infty$) in the curvature example. Edge cases might be set aside when deriving expressions "on paper", but their occurrence in computer code, however rare, must be caught to avoid crashes in simulations.