ARTICLE IN PRESS

Computer Physics Communications ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Computer Physics Communications

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

An object oriented Python interface for atomistic simulations☆

T. Hynninen [a,b,*], L. Himanen [a], V. Parkkinen [c], T. Musso [a], J. Corander [c], A.S. Foster [a]

[a] COMP, Department of Applied Physics, Aalto University School of Science, FI-00076 Aalto, Finland

[b] Department of Physics and Astronomy, University of Turku, FI-20014 Turku, Finland

[c] Department of Mathematics and Statistics, University of Helsinki, FI-00014, Finland

ARTICLE INFO

Article history:
Received 13 March 2015
Accepted 3 September 2015
Available online xxxx

Keywords:
Atomistic simulations
Classical potential
Object oriented
Python
Fortran

ABSTRACT

Programmable simulation environments allow one to monitor and control calculations efficiently and automatically before, during, and after runtime. Environments directly accessible in a programming environment can be interfaced with powerful external analysis tools and extensions to enhance the functionality of the core program, and by incorporating a flexible object based structure, the environments make building and analysing computational setups intuitive. In this work, we present a classical atomistic force field with an interface written in Python language. The program is an extension for an existing object based atomistic simulation environment.

Program summary

Program title: Pysic

Catalogue identifier: AEYE_v1_0

Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEYE_v1_0.html

Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland.

Licensing provisions: Standard CPC licence, http://cpc.cs.qub.ac.uk/licence/licence.html

No. of lines in distributed program, including test data, etc.: 74.743.

No. of bytes in distributed program, including test data, etc.: 758.903.

Distribution format: tar.gz

Programming language: Python, Fortran 90.

Computer: Program has been tested on Linux and OS X workstations, and a Cray supercomputer.

Operating system: Linux, Unix, OS X, Windows.

RAM: Depends on the size of system.

Classification: 7.7, 16.9, 4.14.

External routines: Atomic Simulation Environment, NumPy necessary. Scipy, Matplotlib, HDF5, h5py recommended. The random number generator, Mersenne Twister, is included from the source: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/FORTRAN/mt95.f90

Nature of problem: Automated simulation control, interaction tuning and an intuitive interface for running atomistic simulations.

Solution method: Object oriented interface to a flexible classical potential.

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

* Corresponding author at: Department of Physics and Astronomy, University of Turku, FI-20014 Turku, Finland.
E-mail address: teemu.hynninen@utu.fi (T. Hynninen).

http://dx.doi.org/10.1016/j.cpc.2015.09.010
0010-4655/© 2015 Elsevier B.V. All rights reserved.

Please cite this article in press as: T. Hynninen, et al., An object oriented Python interface for atomistic simulations, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.09.010

Additional comments:
User guide: http://thynnine.github.io/pysic/
Running time: Depends on the size of system.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

Traditionally, the focus in the development of scientific codes has been on the speed, accuracy, and algorithmic functionality, as these are the most important factors deciding the computation cost, reliability, and versatility of the simulations. User friendliness and the flexibility of the interface are often not key concerns. Focus on computational speed usually means profound code optimization specifically for certain types of calculations — sometimes even for specific hardware — and this naturally limits the amount of control the user can be given.

Codes with more emphasis on accessibility have emerged during the last decade. In general, user interface design aims at making it easy and efficient for the user to interact with software. For scientific codes, accessibility typically implies that it is easy, even intuitively so, for the user to build, control, and analyse the simulations. Building is easy when simulation components can be placed freely; control is easy if choosing how to run or constrain the dynamics of the system is straightforward; and analysis is easy if the simulation data can be readily extracted and fed to other tools. In all these aspects, accessibility implies that (i) there are plenty of options so that the user can pick the optimal methods, (ii) the commands for communicating user choices to the program and extracting results are, at least mostly, understandable without an external manual, and (iii) the program can also communicate with other programs and operations can be automated.

One strategy for achieving all of these aspects of accessibility is to construct a programming interface for the code, and Python has become a popular interface language, as it has both powerful scripting capability as well as advanced features of object oriented programming. As an interpreted language it is slow to execute, but it is possible to implement the computationally intensive parts in more efficient compiled languages such as C or Fortran in order to gain back computational speed.

A major advantage of an object oriented interface is that it structures information in a format which humans can understand and manipulate. Parameters have understandable names and objects ideally have intuitive connections. Python can be run in an interactive mode, and there the in-code documentation and query tools allow the user to find the proper keywords even without a manual. Besides streamlining the setup of calculations and minimizing the risk of errors, an intuitive user interface makes it easier for newcomers to start using the program and understand the working principles. This can make the code a tool for both research and teaching.

In this work, we present a Python library for evaluations of classical atomistic force fields. The library is designed to work with the Atomistic Simulation Environment (ASE) [1], an established Python framework for atomistic and electron structure calculations. ASE follows the paradigm of object oriented programming to wrap simulation entities such as atomic structures or dynamics algorithms in Python objects, which are easy to manipulate by both human users and script.

Although ASE provides tools for building and evolving atomic structures, it relies on external programs to determine the interactions between the atoms. Interfaces exist to several such codes, called *calculators* in ASE, at classical (e.g., LAMMPS [2]) and density functional theory level (e.g., GPAW [3]). Our library, *Pysic*, is also a calculator for ASE at the classical level, but instead of providing just a Python interface to an external calculator, Pysic reduces atomistic potentials to Python objects allowing the user to build the interaction model from components. Pysic is not concerned with constructing an atomic structure or its dynamic evolution, as these are already handled in ASE. Instead Pysic calculates the energies and forces of a given structure, i.e., it defines the potential energy surface of the system.

For instance, Pysic describes local pair and many body potentials with `Potential` objects (see Sections 2.1 and 3.1.2). Many body bond order factors can be added as `BondOrderParameters` objects (Sections 2.2 and 3.1.3). Standard Ewald summation is supported and accessed through the `CoulombSummation` object (Sections 2.4 and 3.1.5). Charge dynamics can be controlled using the `ChargeRelaxation` object (Sections 2.5 and 3.1.6). The complete potential, which may be passed to ASE for dynamical simulations, is contained in the `Pysic` object (Section 3.1.1).

2. Functionality

2.1. Local potentials

A library of pair and many body potentials are included in Pysic and also tabulated potentials can be used. The preprogrammed potentials range from simple harmonic springs to elaborate bond order potentials such as the Tersoff potential [4]. Potentials can be targeted to atoms based on their element (such as 'C' or 'H'), their index (a unique number for each atom), or a tag (a numeric label, which can be the same for a group of atoms).

By default, all local potentials are truncated at a cutoff distance specified by the user. However, this may lead to discontinuities in energy and forces and numeric noise. To counter this, smooth cutoffs can be used, where the potentials are multiplied by a function decaying to zero at the cutoff, $\tilde{V}(r) = f(r)V(r)$. The cutoff function used in Pysic is

$$f(r) = \begin{cases} 1, & r \leq r_{\text{soft}} \\ \frac{1}{2}\left(1 + \cos \pi \frac{r - r_{\text{soft}}}{r_{\text{hard}} - r_{\text{soft}}}\right), & r_{\text{soft}} < r \leq r_{\text{hard}} \\ 0, & r > r_{\text{hard}} \end{cases}. \quad (1)$$

This cutoff can also be applied to potentials that do not decay as a function of distance to ensure finite bond lengths.

2.2. Bond order and density-like potentials

Bond order potentials are typically of the type

$$U = \sum_{(i,j)} b_{ij} u_{ij}, \quad (2)$$

where u_{ij} is a pair potential defined, and b_{ij} is the bond order factor. Analogously for single atom potentials,

$$U = \sum_i b_i u_i. \quad (3)$$

Please cite this article in press as: T. Hynninen, et al., An object oriented Python interface for atomistic simulations, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.09.010

Download English Version:

https://daneshyari.com/en/article/6919675

Download Persian Version:

https://daneshyari.com/article/6919675

Daneshyari.com