Computer Physics Communications 198 (2016) 230–237

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 30 September 2015

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

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

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,

$$U = \sum_{i} b_i u_i. \quad (3)$$

This is different from being just a multiplication of two pair potentials by the fact that the bond order factor actually depends on other atoms besides i and j, i.e., it is a many body factor typically of the form (for a pair potential u_{ij})

$$b_{ij} = \frac{1}{2}(\tilde{b}_{ij} + \tilde{b}_{ji}), \tag{4}$$

$$\tilde{b}_{ij} = s_{ij}\left(\sum_k c_{ijk}\right), \tag{5}$$

where s_{ij} is a scaling function and $\sum_k c_{ijk}$ is a sum of atomic triplets including the bond between atoms i and j. This means that the pair interaction of two atoms is modified according to the surroundings of the atoms. For instance, if an atom is overcoordinated, the bond order factor can effectively weaken the bonds the atom forms to make sure increasing the number of bonds an atom forms is not always favoured. Also effects such as preferred relative orientation of the bonds can be included in the bond order factor. Physically this can be interpreted to represent the effects of atomic valence and orbital hybridization, which dictate the optimal number and configuration of covalent bonds an atom likes to form.

For a single atom, the potential has the form

$$b_i = s_i\left(\sum_k c_{ik}\right), \tag{6}$$

which can be thought of as the measure of the atom's coordination or local density. Such potentials are used, for instance, to represent many body effects in delocalized metallic bonding.

A classic example of a bond order factor is the Tersoff potential [4], which describes the fourfold coordinated bonding structure of silicon. It is defined as

$$\tilde{b}_{ij} = \left[1 + \left(\beta_j \sum_{k\neq i,j} \xi_{ijk} g_{ijk}\right)^{\eta_j}\right]^{-\frac{1}{2\eta_j}} \tag{7}$$

$$\xi_{ijk} = f(r_{ij}) \exp\left[a^\mu (r_{ij} - r_{jk})^\mu\right] \tag{8}$$

$$g_{ijk} = 1 + \frac{c_j^2}{d_j^2} - \frac{c_j^2}{d_j^2 + (h_j - \cos\theta_{ijk})^2}. \tag{9}$$

Here f is a cutoff function (1) and θ_{ijk} is the geometric angle defined by the atomic triplet i-j-k.

The Sutton–Chen potential [5] is an example of a density-like potential, where the energy of an atom is directly related to its neighbour density. The potential contains the term

$$U = -c\sum_i \sqrt{\rho_i} \tag{10}$$

$$\rho_i = \sum_j \left(\frac{a}{r_{ij}}\right)^m, \tag{11}$$

and this is reproduced in bond order formalism, Eqs. (3) and (6), with

$$u_i = -c \tag{12}$$

$$s_i(x) = \sqrt{x} \tag{13}$$

$$c_{ik} = \left(\frac{a}{r_{ik}}\right)^m. \tag{14}$$

2.3. Combined potentials

Most of the potentials in the library of interactions offered by Pysic are mathematically simple and often even not very useful on their own. For instance, there are potentials which only depend on the charges of a pair of atoms (i, j), $u_{ij}(q_i, q_j) = q_i^{n_1} q_j^{n_2}$, not their separation. However, it is possible to combine these potentials to form more complicated descriptions. For instance, declaring several potentials u_{ij}^s affecting the same atoms (i, j) (here s is an index for the list of potentials) will simply sum the potentials

$$U = \sum_{(i,j)}\sum_s u_{ij}^s. \tag{15}$$

(Similarly for 1-body, 3-body etc.) It is also possible to add bond order factors b_{ij} and cutoff functions f_{ij}, and multiply the potentials to obtain

$$U = \sum_{(i,j)}\sum_s \prod_p f_{ij}^{s,p} b_{ij}^{s,p} u_{ij}^{s,p}. \tag{16}$$

For example, a charge dependent potential $u_{ij}^1(q_i, q_j) = q_i q_j$ and a distance dependent potential $u_{ij}^2(r_{ij}) = \varepsilon/r_{ij}$ can be combined to form $U(r_{ij}, q_i, q_j) = u_{ij}^1(q_i, q_j) u_{ij}^2(r_{ij}) = \varepsilon q_i q_j / r_{ij}$. An explanation of how this is done in the program is given in Section 3.1.4.

Once a potential (16) has been defined, the calculation of forces is handled automatically.

$$\mathbf{F}_\alpha = -\nabla_\alpha U \tag{17}$$

$$= -\sum_{(i,j)}\sum_s \nabla_\alpha \left(\prod_p f_{ij}^{s,p} b_{ij}^{s,p} u_{ij}^{s,p}\right) \tag{18}$$

$$= -\sum_{(i,j)}\sum_s\sum_p \nabla_\alpha (f_{ij}^{s,p} b_{ij}^{s,p} u_{ij}^{s,p}) \prod_{q\neq p} (f_{ij}^{s,q} b_{ij}^{s,q} u_{ij}^{s,q}), \tag{19}$$

where

$$\nabla_\alpha (f_{ij}^{s,p} b_{ij}^{s,p} u_{ij}^{s,p}) = (\nabla_\alpha f_{ij}^{s,p}) b_{ij}^{s,p} u_{ij}^{s,p} \tag{20}$$

$$+ f_{ij}^{s,p} (\nabla_\alpha b_{ij}^{s,p}) u_{ij}^{s,p} \tag{21}$$

$$+ f_{ij}^{s,p} b_{ij}^{s,p} (\nabla_\alpha u_{ij}^{s,p}). \tag{22}$$

Typically the cutoffs f_{ij} are only a function of the distance between atoms i and j, as are pair potentials u_{ij} (similarly 3-body potentials u_{ijk} depend on the positions of the atoms i, j, k and so on). This means differentiation with respect to atom α gives non-zero contributions to (20)–(22) only when α is either i or j. However, this is not the case for bond order factors. For a factor described by Eq. (5), the gradient is

$$\nabla_\alpha b_{ij} = \frac{1}{2}(\nabla_\alpha \tilde{b}_{ij} + \nabla_\alpha \tilde{b}_{ji}) \tag{23}$$

$$\nabla_\alpha \tilde{b}_{ij} = s'_{ij}\left(\sum_k c_{ijk}\right)\sum_l \nabla_\alpha c_{ijl}. \tag{24}$$

Since the index l sums over all neighbours of atoms i and j, the gradients $\nabla_\alpha c_{ijl}$ are in general non-zero for all the atoms in this neighbourhood, i.e., bond order factors describe true many body interactions. The physical interpretation is that when the presence of an external atom l affects the strength of the bond between atoms i and j, this leads to an effective interaction between the external atom and the bonding pair.

2.4. Coulomb interaction

Direct summation of potentials decaying at the rate $1/r$, such as the Coulomb potential, is only possible for finite systems. In periodic systems, which extend to infinity, the sum

$$E = \sum_{(i,j)} \frac{1}{4\pi\varepsilon_0}\frac{q_i q_j}{r_{ij}} \tag{25}$$

converges only conditionally.

Pysic implements the standard Ewald summation method [6] for evaluating the infinite sum (25). This technique is based on dividing the charge density $\rho(\mathbf{r}) = \sum_i q_i \delta(\mathbf{r} - \mathbf{r}_i)$ in parts for which the sum (25) can be efficiently calculated in by either real or reciprocal space integration.

The split

$$\rho(\mathbf{r}) = \rho_s(\mathbf{r}) + \rho_l(\mathbf{r}) \tag{26}$$

$$\rho_s(\mathbf{r}) = \sum_i [q_i \delta(\mathbf{r} - \mathbf{r}_i) - q_i G_\sigma(\mathbf{r} - \mathbf{r}_i)] \tag{27}$$

$$\rho_l(\mathbf{r}) = \sum_i q_i G_\sigma(\mathbf{r} - \mathbf{r}_i) \tag{28}$$

$$G_\sigma(\mathbf{r}) = \frac{1}{(2\pi\sigma^2)^{3/2}} \exp\left(-\frac{|\mathbf{r}|^2}{2\sigma^2}\right) \tag{29}$$

is used in Pysic. Here, G_σ are Gaussian functions which screen the point charges q_i in real space. The subscript s denotes short ranged and l long ranged interactions.

Using this split, the energy of the system can be written as

$$E = E_s + E_l + E_c, \tag{30}$$

where the components E_s and E_l are the short and long ranged parts of the interaction, respectively, and E_c is a possible correction for non-zero total charge.

The short ranged part of the energy is calculated in real space

$$E_s = \frac{1}{4\pi\varepsilon_0} \int \frac{\rho_s(\mathbf{r})\rho_s(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} \mathrm{d}^3 r \mathrm{d}^3 r' \tag{31}$$

$$= \frac{1}{4\pi\varepsilon_0} \sum_{(i,j)} \frac{q_i q_j}{r_{ij}} \mathrm{erfc}\left(\frac{r_{ij}}{\sigma\sqrt{2}}\right) - \frac{1}{4\pi\varepsilon_0} \frac{1}{\sqrt{2\pi}\sigma} \sum_i^N q_i^2, \tag{32}$$

and the complementary error function $\mathrm{erfc}(r) = 1 - \mathrm{erf}(r) = 1 - \frac{2}{\sqrt{\pi}} \int_0^r e^{-t^2/2} \mathrm{d}t$ makes the sum converge rapidly as $r_{ij}/\sigma \to \infty$. Physically this means that examined from afar, a point charge is completely covered by the screening density and has the apparent total charge of zero. Here, the first sum represents the Coulomb interaction between different point charges and their screening charges, and it is carried over all pairs in the infinite system, but truncated at some reasonable separation $r_{ij} < r_{\mathrm{cut}}$. The second sum represents the self-interaction between a point charge and the Gaussian charge density which screens it. This sum is over all the atoms in the simulation box (N atoms in total) and adds a constant shift in energy.

The long ranged part of the energy,

$$E_l = \frac{1}{4\pi\varepsilon_0} \int \frac{\rho_l(\mathbf{r})\rho_l(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} \mathrm{d}^3 r \mathrm{d}^3 r' \tag{33}$$

cannot be calculated in real space since the screening densities in ρ_l appear as point charges when viewed from a distance. However, since ρ_l is smooth, a Fourier transform can be applied and the sum (33) can be shown to equal

$$E_l = \frac{1}{2V\varepsilon_0} \sum_{\mathbf{k}\neq 0} \frac{e^{-\sigma^2 k^2/2}}{k^2} |S(\mathbf{k})|^2 \tag{34}$$

$$S(\mathbf{k}) = \sum_i^N q_i e^{i\mathbf{k}\cdot\mathbf{r}_i}, \tag{35}$$

where the $\mathbf{k}$-sum is over the vectors of the reciprocal lattice. Again, the sum is truncated at some sufficiently large value of $|\mathbf{k}|$. It is common to truncate the sum over $\mathbf{k}$-vectors simply by changing to a finite sum $\sum_{\mathbf{k}\neq 0} \to \sum_{k_x=-n_x}^{n_x} \sum_{k_y=-n_y}^{n_y} \sum_{k_z=-n_z, \mathbf{k}\neq 0}^{n_z}$, but Pysic uses a spherical truncation $\sum_{\mathbf{k}\neq 0} \to \sum_{\mathbf{k}\neq 0, k<k_{\mathrm{cut}}}$. The function $S(\mathbf{k})$ is the structure factor, and it is calculated by summing over all the atoms in the simulation supercell.

Note that different ways to parameterize these screening functions are used in the literature. In the formulation used here, a large σ corresponds to wide screening functions, meaning fast reciprocal convergence, small k_{cut}, but slow real space convergence, large r_{cut}. Vice versa for a small σ one needs a large k_{cut} and small r_{cut}.

If there is a net charge in the system, the Coulomb energy is infinite. This problem can be circumvented by assigning a uniform, neutralizing background charge density in the simulation volume V and an additional energy component is associated with the interaction of the point charges and this background charge

$$E_c = -\frac{\sigma^2}{4V\varepsilon_0} \left|\sum_i q_i\right|^2. \tag{36}$$

Coulombic forces are obtained as the gradient of the energy $\mathbf{F}_\alpha = -\nabla_\alpha E = -\nabla_\alpha E_s - \nabla_\alpha E_l$, where α denotes the atom affected by the force. We get

$$-\nabla_\alpha E_s = \frac{q_\alpha}{4\pi\varepsilon_0} \sum_j q_j \left[\mathrm{erfc}\left(\frac{r_{\alpha j}}{\sigma\sqrt{2}}\right) \frac{1}{r_{\alpha j}^2} + \frac{1}{\sigma}\sqrt{\frac{2}{\pi}} \exp\left(-\frac{r_{\alpha j}^2}{2\sigma^2}\right) \frac{1}{r_{\alpha j}}\right] \hat{r}_{\alpha j}, \tag{37}$$

where $\hat{r}_{\alpha j} = \mathbf{r}_{\alpha j}/r_{\alpha j}$ is the unit vector pointing from atom α to atom j, and

$$-\nabla_\alpha E_l = -\frac{1}{2V\varepsilon_0} \sum_{\mathbf{k}\neq 0} \frac{e^{-\sigma^2 k^2/2}}{k^2} 2\mathrm{Re}[S^*(\mathbf{k}) \nabla_\alpha S(\mathbf{k})] \tag{38}$$

$$\nabla_\alpha S(\mathbf{k}) = q_\alpha \mathbf{k}(-\sin \mathbf{k}\cdot\mathbf{r}_\alpha + i \cos \mathbf{k}\cdot\mathbf{r}_\alpha). \tag{39}$$

2.5. Dynamic atomic charges

Pysic treats atomic charges as points, although one can also introduce extra charges as dummy atoms. More importantly, it is possible to allow the magnitude of these charges to change dynamically during the simulation to simulate charge redistribution processes.

Assigning an effective inertia M_q to the point charges, the system of atomic positions $\mathbf{r}_i$ and charges q_i can be described with the Lagrangian [7]

$$L(\{\dot{q}\}, \{\dot{\mathbf{r}}\}, \{q\}, \{\mathbf{r}\}) = \sum_i \frac{1}{2} m_i \dot{\mathbf{r}}_i^2 + \sum_i \frac{1}{2} M_q \dot{q}_i^2 - U(\{q\}, \{\mathbf{r}\}) - \nu \sum_i q_i, \tag{40}$$

where the last term is a Lagrange multiplier corresponding to the constraint of fixed total charge $\sum_i q_i = Q$.

The dynamics of this system are given by the equations of motion

$$m_i \ddot{\mathbf{r}}_i = -\nabla_i U \tag{41}$$

$$M_q \ddot{q}_i = -\frac{\partial U}{\partial q_i} - \nu. \tag{42}$$

The "force" acting on charge q_i, $\chi_i = -\partial U/\partial q_i$, is the electropositivity (inverse electronegativity) of atom i. The multiplier ν that conserves the total charge equals the average electropositivity

$$\nu = \bar{\chi} = -\frac{1}{N} \sum_i \frac{\partial U}{\partial q_i}, \tag{43}$$

which is seen by calculating the second derivative of the total charge

$$\ddot{Q} = \frac{1}{NM_q}\sum_i \ddot{q}_i = \frac{1}{N}\sum_i \left(-\frac{\partial U}{\partial q_i} - \nu\right) = \frac{1}{N}\sum_i \chi_i - \bar{\chi} = 0. \quad (44)$$

Similarly, the charges of subsystems such as molecules can be constrained by equalling ν_i with the average electropositivity of the molecule the atoms is a part of. Substituting (43) in (42) leads to the equations of motion for charge

$$M_q\ddot{q}_i = -\frac{\partial U}{\partial q_i} - \bar{\chi} = \Delta\chi. \quad (45)$$

The physical interpretation of (45) is that sites that are more electropositive than average attract positive charge, and those that are more electronegative than average attract negative charge. As these equations do not limit the local charges in any way — only the total charge is conserved — it is possible to drive the system towards having atoms with infinite positive and negative charges. To prevent this, additional restrictions such as local charge dependent potentials should always be included.

Instead of energy conserving charge dynamics, it may be desirable to instead minimize the total energy of the system with respect to the charge distribution. In Pysic, this can be done either by damping the charge dynamics

$$M_q\ddot{q}_i = \Delta\chi - \eta\dot{q}_i, \quad (46)$$

where η is a damping factor, or through a constrained sequential least squares programming algorithm using SciPy optimization routines [8].

It is also possible to run the charge dynamics with a potentiostat instead of the constant charge constraint. Physically this corresponds to connecting the simulation to an external electrode, which fixes the electrostatic potential of the system, but allows charge to flow in or out of the system. The condition for equilibrium with an external potential is that the electropositivity of each atom should equal the constraining potential $\chi_i = \Phi$, so that there is no energy change if charge is brought in or taken out of the system. For a dynamic simulation, this corresponds to the equation of motion (cf. (45))

$$M_q\ddot{q}_i = -\frac{\partial U}{\partial q_i} - \Phi. \quad (47)$$

2.6. Hybrid calculations

The modularity of the potential extends beyond the internal potentials of Pysic. It is possible to use Pysic as a wrapper for other calculators and analyse the system using various descriptions. For instance, one may wish to run calculations at the DFT level but impose constraint forces on the system with Pysic. It is similarly possible to cut the system into domains and apply different methods on the various parts in order to run a quantum mechanics–molecular mechanics (QM/MM) hybrid simulation. In such a simulation, Pysic can act as a filter for dividing the system in quantum mechanical and classical regions and passing these subsystems to other calculators. The classical domain can naturally also be directly handled in Pysic itself.

For creating hybrid systems, Pysic provides a special `HybridCalculator` object. With it one can divide atomic structures into any number of subsystems and define energetic interactions between them. A subsystem is represented by a `Subsystem` object which has a unique name for identification, indices or a tag specifying a certain subset of atoms from the full atomic structure and an ASE compatible calculator used for the atoms. Interactions between the subsystems are defined with `Interaction` objects and any of Pysic's potentials can be used to energetically connect the subsystems. After the subsystems and interactions have been defined, one can use the `HybridCalculator` like any regular ASE Calculator.

There are many strategies for dealing with Coulomb interactions between QM and MM subsystems, and Pysic uses the mechanical embedding scheme [9], i.e. the interactions are modelled at the MM level. Whereas the most basic implementation of mechanical embedding uses parameterized and static charges for the primary QM system, Pysic provides the option of calculating the charges with Bader analysis [10]. In this way no charge parameters are required for the primary system and the dynamic nature of the electronic charge can be taken into account.

To correct the errors produced by defining subsystem boundaries between covalent bonds, Pysic uses the standard hydrogen link atom approach [9]. The hydrogen link atoms can be defined in the `Interaction` object and they are added to the QM subsystem to correct its electronic structure. More advanced approaches to the interface between the QM and MM subsystems are in development.

3. Program structure

3.1. Python classes

Pysic has an object oriented interface, which means the user should build an interaction model from objects represented by Python classes. For instance, simple interactions are build from `Potential` objects, which may be modified with `BondOrderParameters` and extended with a `Coulomb Summation` algorithm. The complete model must be wrapped in a `Pysic` object, which acts as the interface to the ASE library or other programs.

3.1.1. Pysic class

`Pysic` is the class which interfaces with external scripts and libraries making it the class at the top of the class hierarchy. That is, it implements the methods required by an ASE calculator, such as `get_forces()` and `get_potential_energy()`, which invoke the evaluation of these quantities. The most basic use of Pysic involves attaching the calculator to an atomic structure defined in ASE (`ase.Atoms`) and including a list of `Potential` objects to define the interactions.

3.1.2. Potential class

The user can define a simple pair or many body potential as a `Potential` object including the type, parameterization, and affected atoms for the interaction. For instance, a Lennard-Jones potential $U(r) = \varepsilon[(\sigma/r)^{12} - (\sigma/r)^6]$ between two He atoms would be defined as `Potential('LJ', symbols = ['He', 'He'], parameters = [epsilon, sigma])`. Alternatively, one can first create the potential only specifying its type and fully define it using methods such as `set_symbols(['He', 'He'])` and `set_parameter_value('epsilon', 1.0)`. All local potentials also need a cutoff specification to truncate the summation of atomic pairs in a periodic system, as explained in Section 2.1. This is done with methods `set_cutoff()` and `set_cutoff_margin()`.

Typically, a simulation will contain several types of atoms interacting with different potentials. To construct the interaction model for such a system, one defines a separate `Potential` object for each pair or many body interaction and all of them are given to

a Pysic object to wrap them together. This is done with methods pysic.add_potential() or pysic.set_potentials().

3.1.3. BondOrderParameters and Coordinator classes

Bond order factors and density-like potentials, as described in Section 2.2, are defined using BondOrderParameters objects. These contain the classification of a bond order factor similarly to how a Potential defines a potential. Also analogously to the way a calculator object Pysic can contain several Potential objects in order to sum them, a Coordinator object can contain several BondOrderParameters. This allows both inclusion of different types of components c_{ijk} and control of the scaling function f_{ij}, as described in Eq. (5).

The class hierarchy for defining bond order factors works as follows: In terms of Eqs. (2) and (5), u_{ij} is described by the Potential while the function s_{ij} and factors c_{ijk} are defined by one or several BondOrderFactors. If factors are calculated with different parameters (say, according to the chemical elements involved), these must be given using separate BondOrderParameters objects. All the bond order parameters affecting the same potential are then wrapped in a Coordinator object, which is attached to the Potential that is to be modified by the bond order factor.

3.1.4. ProductPotential and CompoundPotential classes

Complicated potentials involving products as defined by Eq. (16) are constructed using the ProductPotential class. The multiplication is invoked by wrapping ordinary Potential objects in a ProductPotential, which is passed to a Pysic calculator just as an ordinary Potential object would be.

The CompoundPotential class is the most sophisticated potential class in Pysic. It can be used to join several potentials together in a single object and also include other functionality such as parameter checks on Python level. It is an abstract class in the sense that it only defines the framework for building the potential wrappers, but the subclasses SuttonChenPotential and Comb are also included demonstrating the realization of a complete potential.

3.1.5. CoulombSummation class

The CoulombSummation class controls Ewald summation for calculating the Coulomb interaction in periodic systems (see Section 2.4). The calculation is invoked by attaching the CoulombSummation to a Pysic calculator. The object contains the parameters controlling the summation algorithm, k_{cut}, r_{cut}, and σ. A utility function pysic.interactions.coulomb.estimate_ewald_parameters() can be used for obtaining a first guess for the parameters based on a given value for the real space cutoff r_{cut}, but the convergence should still always be checked by the user. Currently, only the regular Ewald algorithm has been implemented, but the class is designed to represent other variants as well.

Note that the algorithm assumes the system to be periodic in all directions. CoulombSummation should not be used for a finite system—in such a case the electrostatic interaction should be defined directly using Potential objects.

3.1.6. ChargeRelaxation class

Dynamic charges (see Section 2.5) are invoked using the ChargeRelaxation class. It defines the algorithm used as well as the control parameters. It is also possible to set up a chain of relaxation algorithms to run a simulation with a sequence of different parameters or even algorithms.

In order to run charge dynamics, electronegativities need to be evaluated by a Pysic calculator, which needs to be linked to the ChargeRelaxation.There are two ways to do the link. After a one-way link is formed by attaching Pysic to the ChargeRelaxation with the method set_calculator(), the dynamics are invoked with the method charge_relaxation(). This will return the result of the simulation, but it will not affect the atomic system being evaluated nor will the dynamics be invoked when energies or forces are evaluated. If a two-way link is created through the set_relaxation() method in Pysic, charge dynamics are run automatically before every energy or force calculation. In addition, it can be controlled separately whether the charges in the original atomic structure are automatically updated after relaxation or not.

3.1.7. HybridCalculator class

Used to create and perform hybrid calculations. This class is a fully compatible ASE calculator that can divide atomic structures into any number of subsystems and define energetic interactions between them. Subsystems are added with the function add_subsystem() and interactions between subsystems are added with add_interaction().

3.1.8. Subsystem class

The SubSystem class works as an interface for defining subsystems in hybrid simulations. Through it one can define the atoms that belong to the subsystem and the calculator that is used for them. SubSystem objects also provides special options for QM subsystems: one can setup dynamical charge calculation with enable_charge_calculation() and cell optimization with enable_cell_optimization().

One can define the atoms in the subsystem as a list of indices, with a tag or with a special string: “remaining”, which means all the atoms that are not yet assigned to a subsystem.

3.1.9. Interaction class

The Interaction class works as an interface for defining interactions and link atoms between subsystems in hybrid calculations. With the methods add_potential(), set_potentials(), enable_coulomb_potential() and enable_comb_potential() one can define the Pysic potentials that are present between the subsystems. With add_hydrogen_links() one can setup the hydrogen link atoms.

3.1.10. FastNeighborList class

ASE contains the class NeighborList for listing neighbours, but it is a brute force algorithm meant for small systems. Pysic extends this class to FastNeighborSearch by implementing its own $\mathcal{O}(n)$ neighbour search algorithm. The algorithm is based on a combined Verlet list and cell-linked list scheme [11]. First, the simulation volume is divided into subvolumes whose size depends on the range of interactions. Then, for each atom, the volume containing the atom is found. Lastly, for each atom, the atoms located in the same and adjacent subvolumes are examined to see if they are interacting and closer to each other than the range of their interaction. If they are, they are marked in the neighbour lists as a pair of neighbours.

The spatial decomposition algorithm assumes that the range of interactions is at most half the minimum width of the simulation cell. If this is not the case, the program will fall back on the ASE implementation.

3.2. Python–Fortran interface

Pysic is controlled through a Python interface, where interactions and algorithms are built as objects. Some algorithms such as

charge dynamics are evaluated in Python using, e.g., the NumPy and SciPy libraries [12]. However, most of the heavy computation is implemented in MPI (message passing interface) parallel Fortran90 due to the difference in execution speed of the languages. The Fortran routines are linked to the Python interface using the f2py tool, which is part of NumPy [13].

Fortran90 supports custom types but not, e.g., polymorphism, which prevents direct copying of the Python structure in Fortran. Instead, when, for instance, the potential energy must be evaluated, the atomic structure and interaction model defined in Python are replicated in Fortran in a format ready for numeric evaluation. Simultaneously, the Python interface keeps track of the information fed to the Fortran core and makes sure only the necessary information is passed between the two languages.

The Fortran core appears in Python as the module `pysic.pysic_fortran.pysic_interface` containing a library of functions used for accessing the numeric routines. However, in normal circumstances the user should not directly access these functions. They are mostly called from the `Pysic` class as needed.

3.3. Additional utilities

Pysic also features utility modules and links to external libraries, which are not necessary for the operation of core functions such as energy calculations, but may be useful.

3.3.1. MPI parallelization

The heavy numeric routines are MPI parallel, and the parallelization has been implemented on Fortran level. To utilize this parallelization, the Fortran core must be compiled in an MPI compatible environment. Since the parallel algorithms are hidden from the Python interface, no Python MPI libraries are needed for parallel calculations. It is enough to just launch a Python script in an MPI mode.

Although not a full MPI library, the module `pysic.utility.mpi` contains a group of functions allowing direct access to the MPI routines in Fortran from Python. This allows also for MPI programming in Python.

3.3.2. HDF5 archiving

The module `pysic.utility.archive` contains functions for archiving data. The archiving is based on the hdf5 format [14] and the h5py library [15]. This is a hierarchical binary format, where data is stored in a folder-like structure and described with metadata. The module includes some convenience functions for storing and restoring simulation results, as well as some general functions for navigating and writing data in an hdf5 datafile.

3.3.3. Potential energy analysis

An analysis module `pysic.utility.plot` has functions for exploring and plotting the potential energy surface of the system. In practice this means one atom of the system is moved on a line or a plane and the energy or a force component is recorded. The plotting is done in Python using the Matplotlib library [16].

3.3.4. Structural analysis

The modules `pysic.utility.geometry` and `pysic.utility.outliers` include functions for analysing atomic structures, such as measuring distances and angles. They also include algorithms for automated search of different atomic structures in the given system, such as grains or defects.

The module pysic.utility.outliers scores atoms in a system by how representative they are of similar atoms in the same system. Low scores are likely to indicate defects. Two scores are calculated for each atom, the first by analysing bond angles, the second by analysing bond distances. These can be combined to form a single score.

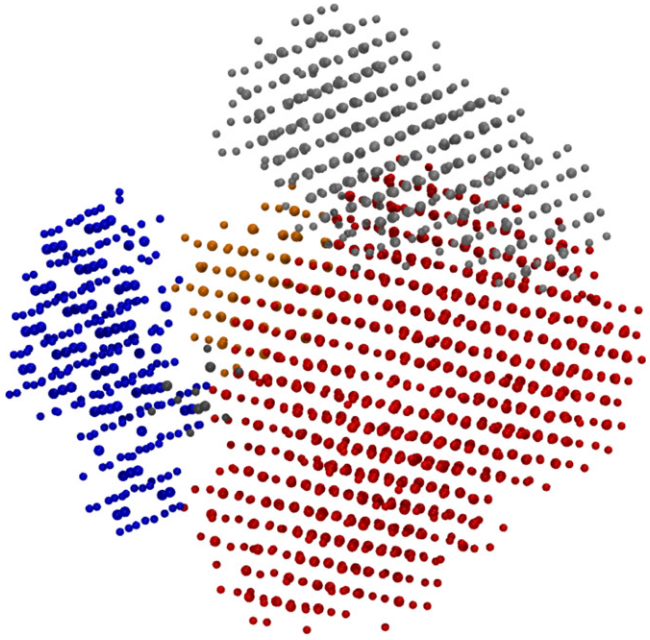

Fig. 1. A HfO_2 cluster consisting of several grains. The structural analysis tools have been used for characterizing individual atoms and subsequently the atoms have been categorized using standard clustering algorithms. The resulting classes have been used for false colouring in the image, revealing the grains.

The score for a particular atom is given by (the logarithm of) the likelihood of observing the bond angles or distances related to the atom. For the purpose of calculating the likelihood, these are assumed to having been generated independently from distributions characterized by the bond angles and distances of similar atoms in the same system.

Automatic search of grains is possible using pysic.utility.outliers. Due to atoms in a grain having a regular structure, the atoms scored highest by the algorithm tend to belong to a grain, if one exists in the data. By only considering some top-scoring proportion of the atoms in a structure, the grain areas will stand out as denser than their surroundings. A density-based clustering algorithm can then be applied to identify the grains. (See Fig. 1).

4. Examples

4.1. Interaction building

The programmable interface allows one to control and tune the interactions through Python scripting. In particular, one can build an algorithm on top of the energy calculator in order to automate the optimization of parameters against any desired observables. This is not limited to parameters either, since the modularity of the program also enables changing the functional form describing the potential at script level.

Also the analysis of force field models benefits from the possibility of breaking down the potential to components. It is possible to pick out the contribution of the different components one at a time, or in groups, and even automatically monitor for situations where a particular part of a potential is especially weak or strong.

As an example, Pysic implements the 2nd generation charge optimized many body (COMB) potentials for silicon and silicon oxide [17]. These potentials are designed to describe the structural

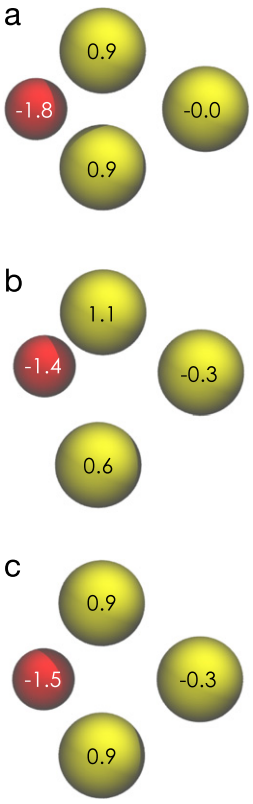

Fig. 2. Si_3O cluster optimized with density functional theory (a), COMB (b) and adjusted COMB (c). The numbers represent the nominal atomic charges, which are obtained using Bader analysis [19] for structure (a) and are a part of the potential for structures (b) and (c).

and mechanical properties of various phase polymorphs of silica, including amorphous silica. They also implement dynamic redistribution of atomic charges. COMB potentials have been parameterized against bulk properties, which means their behaviour for defects and clusters is not guaranteed. Indeed, the default parameterization results in charge instability and asymmetric configurations for some small Si_nO clusters. Fig. 2 shows a Si_3O cluster after structural optimization using (a) the density functional code VASP [18] and (b) the COMB potential. In the latter case, the minimum energy configuration is found when charge is accumulated at one of the Si atoms, leading to an asymmetric configuration.

Using the analysis and scripting power of Pysic, it is fairly easy to search for minimal changes in the potential that restore the expected symmetries. This is a complex problem. For a full reparameterization, the original set of silica polymorphs should also be analysed, and different changes to the interactions may lead to fairly similar results. Fig. 2(c) shows one possibility of tuning the potential involving the change of only one parameter—namely, the attraction between Si and O has been decreased by 10%. This change makes the symmetric configuration shown in the figure the minimum energy structure. In the standard parameterization, it is only a local minimum 0.1 eV higher in energy compared to the structure shown in Fig. 2(b).

4.2. Educational tool

The program has been used as a teaching tool on university level physics courses and summer schools. The interface is simple enough to allow students to build a simulation without prior knowledge of the methodology such as molecular dynamics, yet it is powerful enough to let them to explore the physics and mathematics involved. It is possible to build simulation environments in Python on top of the interface, streamlining the process of constructing a simulation and hiding technicalities not central to the intended learning outcomes. This makes the tool flexible enough to be used both at an advanced level, for teaching computational physics methodology in detail, and at an introductory physics level, where students can essentially build and run computational experiments without knowing all the algorithms that are being used by the software.

As a computational laboratory, the program can support problem-based and enquiry-driven learning practices, where the students are presented with open problems for which no unique answers exist. In such a context, the students need to formulate and test their own hypotheses. If these hypotheses are to be tested via simulations, the simulation environment must be flexible and intuitive enough to allow the students to design and create simulations that produce information on the particular ideas they are examining. Environments built on ASE and Pysic can be made robust yet free.

Acknowledgements

We acknowledge financial support from the Academy of Finland through its Centres of Excellence Programme (Project No. 251748), EU projects MORDRED (Contract No. 261868) and PAMS (Contract No. 610446). T.M. was supported by the Finnish Academy of Science and Letters and Vilho, Yrjö, Kalle Väisälä Foundation. We acknowledge use of the Finnish CSC–IT Center for Sciences supercomputing resources.

References

[1] S.R. Bahn, K.W. Jacobsen, An object-oriented scripting interface to a legacy electronic structure code, Comput. Sci. Eng. 4 (3) (2002) 56–66.
[2] S. Plimpton, Fast parallel algorithms for short-range molecular-dynamics, J. Comput. Phys. 117 (1) (1995) 1–19.
[3] J. Enkovaara, C. Rostgaard, J.J. Mortensen, J. Chen, M. Dułak, L. Ferrighi, J. Gavnholt, C. Glinsvad, V. Haikola, H.A. Hansen, H.H. Kristoffersen, M. Kuisma, A.H. Larsen, L. Lehtovaara, M. Ljungberg, O. Lopez-Acevedo, P.G. Moses, J. Ojanen, T. Olsen, V. Petzold, N.A. Romero, J. Stausholm-Møller, M. Strange, G.A. Tritsaris, M. Vanin, M. Walter, B. Hammer, H. Häkkinen, G.K.H. Madsen, R.M. Nieminen, J.K. Nørskov, M. Puska, T.T. Rantala, J. Schiøtz, K.S. Thygesen, K.W. Jacobsen, Electronic structure calculations with GPAW: a real-space implementation of the projector augmented-wave method, J. Phys.: Condens. Matter. 22 (25) (2010) 253202.
[4] J. Tersoff, New empirical approach for the structure and energy of covalent systems, Phys. Rev. B 37 (12) (1988) 6991–7000.
[5] A.P. Sutton, J. Chen, Long-range Finnis–Sinclair potentials, Phil. Mag. Lett. 61 (3) (2006) 139–146.
[6] A. Toukmaji, J. Board Jr., Ewald summation techniques in perspective: a survey, Comput. Phys. Comm. 95 (1996) 73–92.
[7] S.W. Rick, S.J. Stuart, B.J. Berne, Dynamical fluctuating charge force-fields—application to liquid water, J. Chem. Phys. 101 (7) (1994) 6141.
[8] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, [Online; accessed 2015-01-21] (2001–). URL http://www.scipy.org/.
[9] D. Bakowies, W. Thiel, Hybrid models for combined quantum mechanical and molecular mechanical approaches, J. Phys. Chem. 100 (25) (1996) 10580–10594. http://dx.doi.org/10.1021/jp9536514.
[10] R. Bader, Atoms in Molecules: A Quantum Theory, Clarendon Press, Oxford New York, 1990.
[11] W. Mattson, B. Rice, Near-neighbor calculations using a modified cell-linked list method, Comput. Phys. Comm. 119 (1999) 135–148.
[12] T.E. Oliphant, Python for scientific computing, Comput. Sci. Eng. 9 (3) (2007) 10–20.
[13] P. Peterson, F2PY: a tool for connecting Fortran and Python programs, IJCSE 4 (4) (2009) 296.
[14] The HDF Group, Hierarchical Data Format, version 5.
[15] A. Collette, Python and HDF5, Unlocking Scientific Data, O'Reilly Media, 2013.
[16] J.D. Hunter, Matplotlib: A 2D graphics environment, Comput. Sci. Eng. 9 (3) (2007) 90–95.
[17] T.-R. Shan, D. Bryce, J. Hawkins, A. Asthagiri, S. Phillpot, S. Sinnott, Second-generation charge-optimized many-body potential for Si/SiO_2 and amorphous silica, Phys. Rev. B 82 (23) (2010) 235302.
[18] G. Kresse, J. Furthmuller, Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set, Phys. Rev. B 54 (16) (1996) 11169–11186.
[19] G. Henkelman, A. Arnaldsson, H. Jonsson, A fast and robust algorithm for Bader decomposition of charge density, Comp. Mater. Sci. 36 (3) (2006) 354–360.