TOPICAL REVIEW

The atomic simulation environment—a Python library for working with atoms

IOP Publishing Journal of Physics: Condensed Matter

J. Phys.: Condens. Matter **29** (2017) 273002 (30pp) https://doi.org/10.1088/1361-648X/aa680e

Topical Review

The atomic simulation environment—a Python library for working with atoms

Ask Hjorth Larsen[1,2], Jens Jørgen Mortensen[3], Jakob Blomqvist[4], Ivano E Castelli[5], Rune Christensen[6], Marcin Dułak[3], Jesper Friis[7], Michael N Groves[8], Bjørk Hammer[8], Cory Hargus[9], Eric D Hermes[10], Paul C Jennings[6], Peter Bjerre Jensen[6], James Kermode[11], John R Kitchin[12], Esben Leonhard Kolsbjerg[8], Joseph Kubal[13], Kristen Kaasbjerg[14], Steen Lysgaard[6], Jón Bergmann Maronsson[15], Tristan Maxson[13], Thomas Olsen[3], Lars Pastewka[16], Andrew Peterson[9], Carsten Rostgaard[3,17], Jakob Schiøtz[3], Ole Schütt[18], Mikkel Strange[3], Kristian S Thygesen[3], Tejs Vegge[6], Lasse Vilhelmsen[8], Michael Walter[19], Zhenhua Zeng[13] and Karsten W Jacobsen[3]

[1] Nano-bio Spectroscopy Group and ETSF Scientific Development Centre, Universidad del País Vasco UPV/EHU, San Sebastián, Spain
[2] Dept. de Ciència de Materials i Química Física & IQTCUB, Universitat de Barcelona, c/ Martí i Franquès 1, 08028 Barcelona, Spain
[3] Department of Physics, Technical University of Denmark, Lyngby, Denmark
[4] Faculty of Technology and Society, Malmö University, Sweden
[5] Department of Chemistry, University of Copenhagen, Denmark
[6] Department of Energy Conversion and Storage, Technical University of Denmark, Lyngby, Denmark
[7] SINTEF Materials and Chemistry, Norway
[8] Interdisciplinary Nanoscience Center (iNANO), Department of Physics and Astronomy, Aarhus University, Denmark
[9] School of Engineering, Brown University, Providence, RI, United States of America
[10] Theoretical Chemistry Institute and Department of Chemistry, University of Wisconsin-Madison, WI, United States of America
[11] Warwick Centre for Predictive Modelling, School of Engineering, University of Warwick, United Kingdom
[12] Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA, United States of America
[13] School of Chemical Engineering, Purdue University, West Lafayette, IN, United States of America
[14] Department of Micro- and Nanotechnology, Technical University of Denmark, Lyngby, Denmark
[15] Síminn, Reykjavík, Iceland
[16] Institute for Applied Materials—Computational Materials Science, Karlsruhe Institute of Technology, Germany
[17] Netcompany IT and business consulting A/S, Copenhagen, Denmark
[18] Nanoscale Simulations, ETH Zürich, 8093 Zürich, Switzerland
[19] Freiburg Centre for Interactive Materials and Bioinspired Technologies, University of Freiburg, Germany

E-mail: asklarsen@gmail.com, jensj@fysik.dtu.dk and kwj@fysik.dtu.dk

Received 13 December 2016
Accepted for publication 21 March 2017
Published 7 June 2017

Abstract

The atomic simulation environment (ASE) is a software package written in the Python programming language with the aim of setting up, steering, and analyzing atomistic simulations. In ASE, tasks are fully scripted in Python. The powerful syntax of Python

combined with the NumPy array library make it possible to perform very complex simulation tasks. For example, a sequence of calculations may be performed with the use of a simple 'for-loop' construction. Calculations of energy, forces, stresses and other quantities are performed through interfaces to many external electronic structure codes or force fields using a uniform interface. On top of this calculator interface, ASE provides modules for performing many standard simulation tasks such as structure optimization, molecular dynamics, handling of constraints and performing nudged elastic band calculations.

Keywords: density functional theory, molecular dynamics, electronic structure theory

(Some figures may appear in colour only in the online journal)

1. Introduction

The understanding of behaviour and properties of materials at the nanoscale has developed immensely in the last decades. Experimental techniques like scanning probe microscopy and electron microscopy have been refined to provide information at the sub-nanometer scale. At the same time, theoretical and computational methods for describing materials at the electronic level have advanced and these methods now constitute valuable tools to obtain reliable atomic-scale information [1].

The atomic simulation environment (ASE) is a collection of Python modules intended to set up, control, visualise, and analyse simulations at the atomic and electronic scales. ASE provides Python classes like 'Atoms' which store information about the properties and positions of individual atoms. In this way, ASE works as a *front-end* for atomistic simulations where atomic structures and parameters controlling simulations can be easily defined. At the same time, the full power of the Python language is available so that the user can control several interrelated simulations interactively and in detail.

The execution of many atomic-scale simulations requires information about energies and forces of atoms, and these can be calculated by several methods. One of the most popular approaches is density functional theory (DFT) which is implemented in different ways in dozens of freely available codes [2]. DFT codes calculate atomic energies and forces by solving a set of eigenvalue equations describing the system of electrons. A simpler but also more approximate approach is to use interatomic potentials (or so-called force fields) to calculate the forces directly from the atomic positions [3]. ASE can use DFT and interatomic potential codes as *backends* called 'Calculators' within ASE. By writing a simple Python interface between ASE and, for example, a DFT code, the code is made available as an ASE calculator to the users of ASE. At the same time, researchers working with this particular code can benefit from the powerful setup and simulation facilities available in ASE. Furthermore, the uniform interface to different calculators in ASE makes it easy to compare or combine calculations with different codes. At the moment, ASE has interfaces to about 30 different atomic-scale codes as described in more detail later.

A few historical remarks: in the 1990s, object-oriented programming was widespread in many fields but not used much in computational physics. Most physics codes had a monolithic character written in compiled languages like Fortran or C using static input/output files to control the execution. However, the idea that physics codes should be 'wrapped' in object-oriented scripting languages was put forward [4]. The idea was that the object-oriented approach would allow the user of the program to operate with more understandable 'physics' objects instead of technical details, and that the scripting would encourage more interactive development and testing of the program to quickly investigate new ideas. One of the tasks was therefore also to split up the Fortran or C code to make relevant parts of the code available individually to the scripting language. Also in the mid-nineties, the book on Design Patterns [5] was published discussing how to program efficiently using specific object-oriented patterns for different programming challenges. These patterns encourage better structuring of the code, for example by keeping different sub-modules of the code as independent as possible, which improves readability and simplifies further development.

Inspired by these ideas, the first version of ASE [6] was developed around the turn of the century to wrap the DACAPO DFT code [7] at the Center of Atomic-scale Materials Physics at the Technical University of Denmark. DACAPO is written in Fortran and controlled by a text input file. It was decided to use Python both because of the general gain in popularity at the time—although mostly in the computer science community—and because the development of numerical tools like Numeric and NumArray, the predecessors of NumPy [8], were under way. Gradually, more and more features, like atomic dynamics, were moved from DACAPO into ASE to provide more control at the flexible object-oriented level.

A major rewrite of ASE took place with the release of both versions 2 and 3. In the first version of the code, the 'objectification' was enthusiastically applied, so that for example the position of an atom was an object. This meant that the user applying the 'get position' method to an Atom object would receive such a Position object. One could then query this object to get the coordinates in different frames of reference. Over time, it turned out that too much 'objectification' made ASE more difficult to use, in particular for new users who experienced a fairly steep learning curve to become familiar with the different objects. It was therefore decided to lower the degree of abstraction so that for example positions would be described by simply the three coordinates in a default frame of reference. However, the general idea of creating code consisting of

independent modules by applying appropriate design patterns has remained. One example is the application of the 'observer-pattern' [5], which allows for development of a small module of code (the 'Observer') to be called regularly during a simulation. By just attaching the Observer to the 'Dynamics' object, which is in control of the simulation, the Observer calculations will automatically be performed as requested.

ASE has now developed into a full-fledged international open-source project with developers in several countries. Many modules have been added to ASE to perform different tasks, for example the identification of transition states using the nudged elastic band method [9, 10]. Recently, a database module which allows for convenient storage and retrieval of calculations including a web-interface has also been developed. More calculators are added regularly as backends, and new open-source projects like Amp (Atomistic Machine-learning Package) [11] build on ASE as a flexible interface to the atomic calculators. The refinement of libraries like NumPy allows for more and more tasks to be efficiently performed at the Python level without the need for compiled languages. This also opens up new possibilities for both inclusion of more modules in ASE and for efficient use of ASE in other projects.

2. Overview

In the following we provide a brief overview of the main features of ASE.

2.1. Python

A distinguishing feature of ASE is that most tasks are accomplished by writing and running Python scripts. Python is a dynamically typed programming language with a clear and expressive syntax. It can be used for writing everything from small scripts to large programs or libraries like ASE itself. Python has gained popularity for scientific applications [12–14], thanks particularly to the free and open-source numerical libraries of the SciPy community.

Consider the classical approach of many computational codes, where a compiled binary runs on a specially formatted input file. A single run can perform only those actions that are implemented in the code, and any change would require modifying the source code and recompiling. With ASE, the scripting environment makes it trivial to combine several tasks in any way desired, to attach *observers* that run custom code as callbacks during longer simulations, or to customize calculation outputs.

Here is a simple example showing an interactive session in the Python interpreter:

```
>>> from ase import Atoms
>>> from ase.optimize import BFGS
>>> from ase.calculators.nwchem import NWChem
>>> from ase.io import write
>>> h2 = Atoms('H2',
...            positions=[[0, 0, 0],
...                       [0, 0, 0.7]])
...
>>> h2.calc = NWChem(xc='PBE')
>>> opt = BFGS(h2)
>>> opt.run(fmax=0.02)
BFGS:    0  19:10:49     -31.435229      2.2691
BFGS:    1  19:10:50     -31.490773      0.3740
BFGS:    2  19:10:50     -31.492791      0.0630
BFGS:    3  19:10:51     -31.492848      0.0023
>>> write('H2.xyz', h2)
>>> h2.get_potential_energy()
-31.492847800329216
```

This example defines an ASE `Atoms` object representing a hydrogen molecule with an approximate 0.7 Å bond length. ASE uses eV and Å as units. The molecule is equipped with a *calculator*, `NWChem`, which is the ASE interface to the NWChem [15, 16] code. It is instructed to use the PBE functional [17] for exchange and correlation effects. Next, a structure optimization is performed using the BFGS [18] algorithm as implemented within ASE. The following lines of output text show energy and maximum force for each iteration until it converges.

Note that due to dynamic typing, it is not necessary to declare the types of variables, and due to automatic memory management, there are no explicit allocations or deallocations. This makes the language clear and concise. However, Python itself is not designed for heavy numerical computations. High-performance computational codes would need to be written in a language that gives more control over memory, such as C or Fortran. Several Python libraries are available which provide efficient implementations of numerical algorithms and other scientific functionality. ASE relies on three external libraries:

- NumPy [8] provides a multidimensional array class with efficient implementations of basic arithmetic and other common mathematical operations for ordinary dense arrays, such as matrix multiplication, eigenvalue computation, and fast Fourier transforms.
- SciPy [19] works on top of NumPy and provides algorithms for more specialized numerical operations such as integration, optimization, special mathematical functions, sparse arrays, and spline interpolation.
- matplotlib [20] is a plotting library which can produce high-quality plots of many types.

Together, the three libraries provide an environment reminiscent of applications such as Octave or Matlab.

It is also possible to write *extensions* in C that can be called from Python, or to link to compiled libraries written in another language. The DFT code GPAW [21, 22], which is designed specifically to work with ASE, consists of about 85–90% Python with the remainder written in C. Almost all logically complex tasks are written in Python, whereas only computationally demanding parts, typically tight loops of floating point operations, are written in C. Like most DFT codes, GPAW also relies on external libraries such as BLAS and LAPACK for high performance. ASE itself, however, does

not perform extremely performance-critical functions and is written entirely in Python.

2.2. Atoms and calculators

At the center of ASE is the `Atoms` object. It represents a collection of atoms of any chemical species with given Cartesian positions. Depending on the type of simulation, the atoms may have more properties such as velocities, masses, or magnetic moments. They may also have a simulation cell given by three vectors and can represent crystals, surfaces, chains, or the gas phase, by prescribing periodic or non-periodic boundary conditions along the directions of each cell vector. `Atoms` objects behave similarly to Python lists:

```
from ase import Atoms
a = Atoms()  # empty
a.extend(Atoms('Xe10'))  # add 10 Xe atoms
a.append('H')  # append hydrogen atom
print(a[0])  # first atom
print(a[1])  # second atom
del a[-3:]  # delete three last atoms
```

Properties of the `Atoms` object are backed by NumPy arrays to retain good performance even with thousands or millions of atoms.

ASE provides modules to generate many kinds of structures such as bulk crystals, surfaces, or nanoparticles, and can read and write a large number of different file formats. Structures can further be manipulated by many operations such as rotations, translations, repetition as a supercell, or simply by modifying the values of the positions array. Complex systems can be formed by combining (adding) several atoms objects. Section 3 gives a detailed description.

Atoms can be equipped with *calculators*. A calculator is a black box that can take atomic numbers and positions from the atoms and calculate the energy and forces, and possibly other properties such as the stress tensor. For example, calling `get_potential_energy()` on the `Atoms` object will trigger a call to the calculator which evaluates and retrieves the energy. What exactly happens behind the scenes depends on the calculator's implementation. Many calculators are interfaces to established electronic structure codes, and in this case, the call will generally result in a self-consistent DFT calculation according to the parameters which were passed when creating the calculator.

There are ASE calculators for many different electronic structure, (semi-)empirical, tight-binding and classical (reactive) interatomic potential codes. Some calculator interfaces are maintained and distributed as a part of ASE, while others are included with the external codes themselves, and a few are distributed by third parties. A few calculators are not just interfaces, but are implemented fully in Python and are included with ASE. This is summarized in table 1.

In addition to the listed calculators, there are two calculators which wrap ordinary calculators and add corrections to the energies and forces: one for the van der Waals corrections by Tkatchenko and Scheffler [49], and one for the Grimme D3 dispersion correction [50–52].

The most common way to communicate with the codes is by reading and writing files, but some have more efficient interfaces that use sockets or pipes, or simply run within the same physical process. This is discussed in section 5.

2.3. Atomistic algorithms in ASE

On top of the atoms–calculator interface, ASE provides algorithms for different tasks in atomistic simulations. These algorithms typically rely on energies and forces evaluated by the calculators, but interact only with the `Atoms` objects, and know nothing about the underlying implementation of the calculator.

- Molecular dynamics with different controls such as thermostats, section 6.1.
- Structure optimization using the atomic forces, section 6.2.
- Saddle-point searches on the potential energy surface, such as determination of minimum-energy paths for a reaction using the nudged elastic band method (section 6.4) or the dimer method (section 6.5).
- Global structure optimization using basin hopping (section 7.1) or minima hopping (section 7.2) algorithms.
- Genetic algorithms for optimization of structure or chemical composition, section 7.3.
- Analysis of molecular vibrational modes or phonon modes for solids, section 8.

These features and more will be discussed in the following sections.

3. Generating structures

The first problem in atomistic simulations is to set up an atomic structure. Using the built-in GUI of ASE, a structure can be built by adding the desired atoms to the system and moving them to the desired location manually. More general structures can be constructed by scripting. This also allows for the specification of other properties such as constraints, magnetic moments and charges of the individual atoms.

3.1. Generic structures

ASE has modules to define a wide range of different structures; nanotubes, bulk lattices, surfaces, and nanoparticles are a few such examples. The simplest predefined structures involve gas phase species and small organic molecules. ASE includes the G2 test set of 148 molecules [53], which are useful as predefined adsorbates for slab calculations. The example below shows the manual definition of H_2 and how to retrieve H_2O from the G2 collection:

```
from ase import Atoms
h2 = Atoms('H2', [(0, 0, 0), (0, 0, 0.74)])

from ase.build import molecule
water = molecule('H2O')
```

Table 1. Summary of ASE calculators.

Code	Link	Reference	Communication
Abinit	www.abinit.org/	[23]	Files
ASAP[a]	https://wiki.fysik.dtu.dk/asap/		Python
Atomistica[a]	https://github.com/Atomistica/atomistica		Python
Castep	www.castep.org/	[24]	Files
CP2K	https://www.cp2k.org/	[25]	Interprocess
deMon	www.demon-software.com/	[26]	Files
DFTB +	www.dftb-plus.info/	[27]	Files
EAM	Part of ASE	[28]	Python
ELK	http://elk.sourceforge.net/		Files
EMT	Part of ASE	[29]	Python
Exciting	http://exciting-code.org/	[30]	Files
FHI-aims	https://aimsclub.fhi-berlin.mpg.de/	[31]	Files
Fleur	www.flapw.de/	[32]	Files
Gaussian	www.gaussian.com/	[33]	Files
GPAW[a]	https://wiki.fysik.dtu.dk/gpaw/	[22]	Python
Gromacs	www.gromacs.org/	[34]	Files
Hotbit[a]	https://github.com/pekkosk/hotbit/	[35]	Python
Dacapo	https://wiki.fysik.dtu.dk/dacapo/	[7]	Interprocess
JDFTx[a]	https://sourceforge.net/projects/jdftx/	[36]	Files
LAMMPS	http://lammps.sandia.gov/	[37]	Files
Lennard–Jones	Part of ASE	[38]	Python
matscipy[a]	https://github.com/libAtoms/matscipy		Python
MOPAC	http://openmopac.net/	[39]	Files
Morse	Part of ASE	[40]	Python
NWChem	www.nwchem-sw.org/	[16]	Files
Octopus	www.tddft.org/programs/octopus/	[41]	Files
OpenKIM[b]	https://openkim.org/	[42]	Python
Quantum Espresso[c]	www.quantum-espresso.org/	[43]	Interprocess
QUIP[a]	http://libatoms.github.io/QUIP/	[44]	Python
SIESTA	http://departments.icmab.es/leem/siesta/	[45]	Files
TIP3P	Part of ASE	[46]	Python
Turbomole	www.turbomole.com/	[47]	Files
VASP	https://www.vasp.at/	[48]	Files

[a] Distributed as part of the code instead of with ASE.
[b] Distributed by third party: https://github.com/mattbierbaum/openkim-kimcalculator-ase
[c] Distributed by third party: https://github.com/vossjo/ase-espresso

Bulk crystals can be constructed manually like this:

```
a = 3.6
cu = Atoms('Cu', [(0, 0, 0)],
           cell=[(0, a / 2, a / 2),
                 (a / 2, 0, a / 2),
                 (a / 2, a / 2, 0)],
           pbc=[True, True, True])
```

or equivalently with the shortcut:

```
from ase.build import bulk
cu = bulk('Cu', 'fcc', a=3.6)
```

3.2. Space group

In three dimensions, the set of all symmetry operations of a crystalline structure is the *space group* for this structure. All symmetry operations of the in total 230 unique space groups are listed in the file `spacegroup.dat` in the `ase.spacegroup` package. The space group numbers and nomenclature follow the definitions in international tables [54].

The spacegroup package can create, and to some extent manipulate, crystalline structures. For users of ASE, the most important function in this package is `crystal()` which returns a unit cell of a crystalline structure given information about the space group, lattice parameters and scaled coordinates of the non-equivalent atoms. The example below shows how to create a unit cell of beryl[20] from crystallographic information typically provided in publications:

```
from ase.spacegroup import crystal

# Adamo et al. Mineralogical Magazine
# 72 (2008) 799-808
beryl = crystal(
```

[20] Beryl is a naturally occurring mineral with chemical composition $Be_3Al_2(SiO_3)_6$ and a hexagonal crystal structure with 58 atoms in the unit cell.

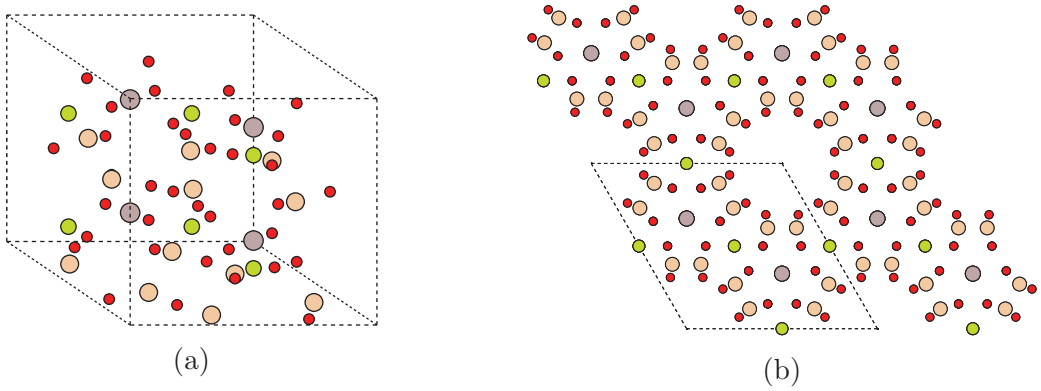


Figure 1. (a) Unit cell of beryl, (b) same cell repeated twice and seen along [0 0 1].

```
    symbols=['Al', 'Be', 'Si', 'O', 'O'],
    basis=[(2. / 3, 1. / 3, 0.25),   # Al
           (0.5,    0.0,    0.25),   # Be
           (0.39,   0.12,   0.00),   # Si
           (0.50,   0.15,   0.15),   # O1
           (0.31,   0.24,   0.00)], # O2
    spacegroup='P 6/m c c',  # no 192
    cellpar=[9.25, 9.25, 9.22,
             90, 90, 120])
```

The resulting structure is shown in figure 1. The `Spacegroup` object can be used to investigate symmetry-related properties of the structure, like whether beryl is centrosymmetric;

```
>>> from ase.spacegroup import Spacegroup
>>> sg = Spacegroup(192)
>>> sg.centrosymmetric
True
```

or to find its non-equivalent scaled atomic positions:

```
>>> rcoord = beryl.get_scaled_positions()
>>> print(sg.unique_sites(rcoord))
[[ 0.33333333  0.66666667  0.25      ]
 [ 0.5         0.          0.25      ]
 [ 0.88        0.27        0.        ]
 [ 0.65        0.15        0.65      ]
 [ 0.24        0.31        0.5       ]]
```

3.3. Surfaces and interfaces

Surfaces are generated by cleaving a bulk material along one of its crystallographic planes. The functions `ase.build.surface()` and `ase.build.cut()` can create arbitrary surface slabs. ASE also has specialized functions to generate many of the common metal surfaces, such as FCC(1 1 1) and HCP(0 0 0 1). Slabs of different sizes and thicknesses can be defined using this tool. For periodic slab models, the vacuum between the slab and the periodic images can also be defined. Molecules can be placed as adsorbates in predefined binding sites, such as top, bridge and hollow, as shown in figure 2(a) where an N_2 molecule is adsorbed on a Pt(1 1 1) surface:

```
# Pt FCC (111) with absorbed N2
from ase.build import (fcc111,
    add_adsorbate, molecule)
slab = fcc111('Pt', size=(4, 4, 4),
    a=4.0, vacuum=6.0)
add_adsorbate(slab, molecule('N2'),
    height=3.0, position='ontop')
add_adsorbate(slab, molecule('N2'),
    height=3.0, offset=(2, 2),
    position='ontop')
```

Both utilizing the GUI and via scripting, single metal nanoparticles can be constructed using the Wulff construction method.

```
# Graphene nanoribbon
from ase.build import graphene_nanoribbon
ribbon = graphene_nanoribbon(2, 2,
    type='armchair', saturated=True)

# 55-atom cuboctahedral gold nanoparticle
from ase.cluster import Octahedron
cluster = Octahedron('Au', 5, cutoff=2)
```

More complicated surfaces and interfaces can be made by combining existing structures or by combining the structure generators with other ASE functions. Here, an FeO film on a Pt(1 1 1) surface is constructed by combining two slabs (figure 3(a)):

```
# FeO film on Pt(111)
from ase.build import (root_surface,
    fcc111, stack)

primitive_substrate = fcc111('Pt',
    size=(1, 1, 4), vacuum=2)
substrate = root_surface(
    primitive_substrate, 84)
primitive_film = fcc111('Fe',
    size=(1, 1, 2), a=3.9, vacuum=1)
primitive_film[1].symbol = 'O'
film = root_surface(primitive_film, 67)
interface = stack(substrate, film,
    fix=0, maxstrain=None)
```

The neighbour list function provides the number and type of atoms in the vicinity of a given atom and can therefore

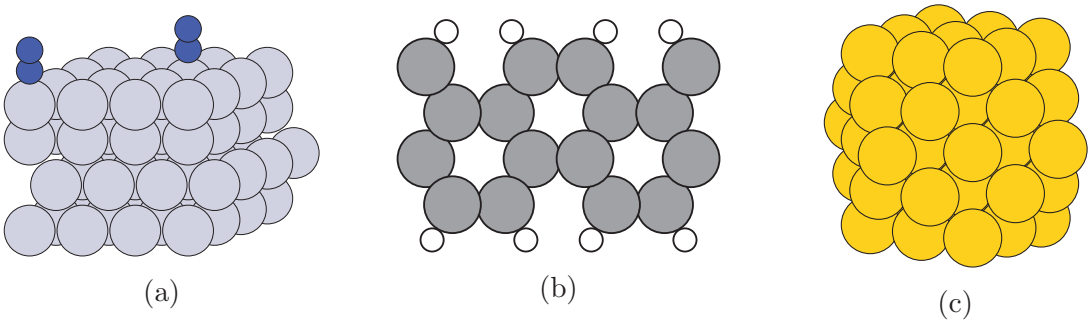


Figure 2. (a) Platinum surface with 2 N_2 adsorbed at top sites [55], (b) carbon nanoribbon with H-saturated edge, and (c) cuboctahedral gold nanoparticle constructed using various functions of ASE.

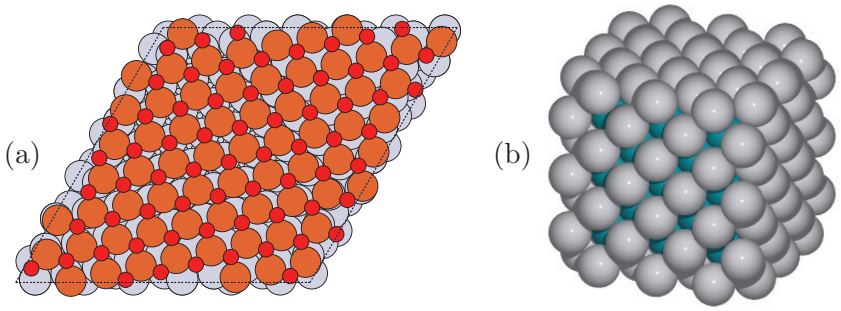


Figure 3. (a) Experimentally observed FeO film supported on Pt(1 1 1) substrate [56], (b) Pd core/shell nanoparticle with Pt surface [57].

be used to find the coordination number of an atom. A Pd/Pt core/shell nanoparticle can be constructed by utilizing the coordination number to identify surface atoms (figure 3(b)):

```
# Pt/Pd Core-shell nanoparticle
from ase.neighborlist import NeighborList
from ase.cluster import FaceCenteredCubic

atoms = FaceCenteredCubic('Pd',
    surfaces=[[1, 0, 0]], layers=[3])
nl = NeighborList([1.4] * len(atoms),
    self_interaction=False, bothways=True)
nl.build(atoms)
for x, atom in enumerate(atoms):
    indices, offsets = nl.get_neighbors(x)
    if len(indices) < 12:
        atom.symbol = 'Pt'
```

4. Reading and writing structures

ASE presently recognizes 65 file formats for storing atomic configurations. Among these is the simple `xyz` format, as well as many less well-known formats. ASE also provides two of its own native file formats [58] in which all the items in an `Atoms` object—including momenta, masses, constraints, and calculated properties such as the energy and forces—can be stored. The two file formats are the compact `traj` format and a larger but human-readable `json` format. The `traj` format is tightly integrated with other parts of ASE such at structure optimizers. A `traj` file can contain one or more `Atoms` objects, but all of them must have the same number and kind of atoms in the same order. The `json` file format can also contain more that one `Atoms` object, but there are no restrictions on what it can contain (see also section 4.1). Reading and writing is done with the `read()` and `write()` functions from the `ase.io` module. The latter can also be used to create images (`png`, `svg`, `eps` and `pdf`). It can also use POVRAY [59] to render the output, like the core/shell nanoparticle on figure 3(b).

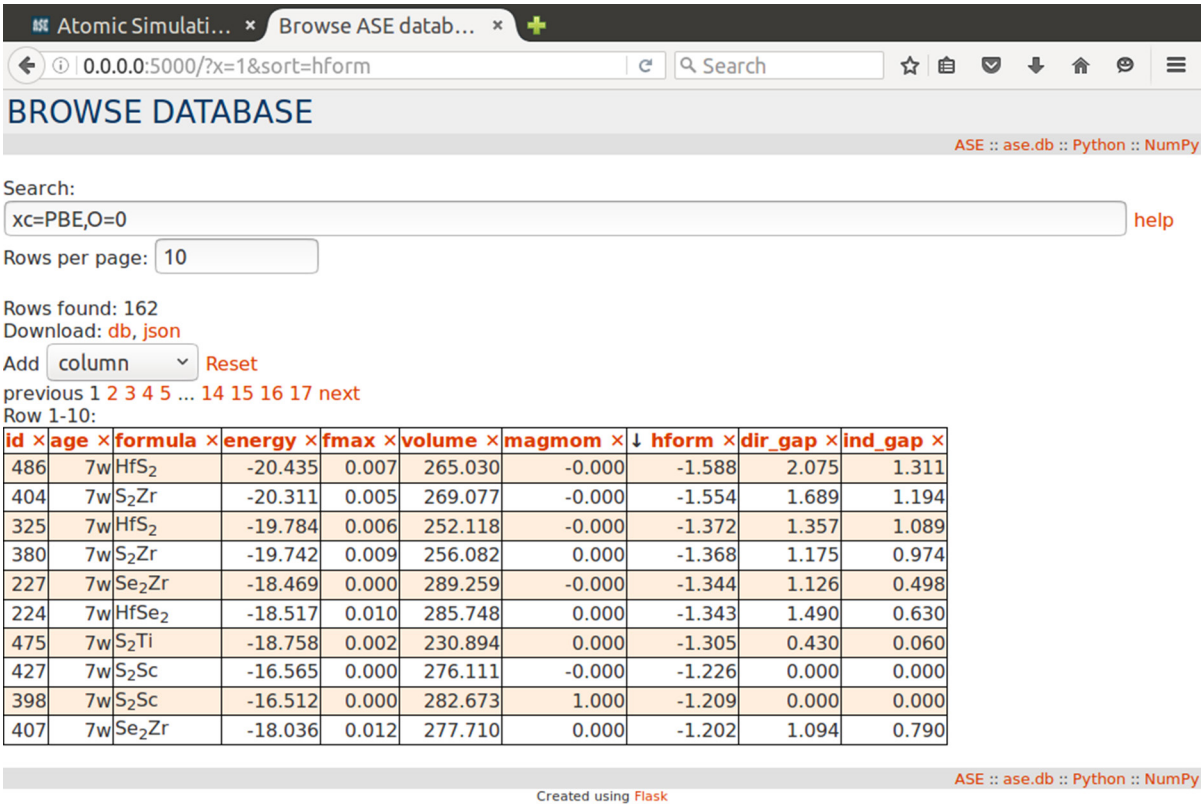
4.1. Database

ASE has a database module (`ase.db`) that can be used for storing and retrieving atoms and associated data in a compact and convenient way. A database for atomic configurations is ideal for keeping systematic track of many related calculations. This will for example be the situation in computational screening studies or when working with genetic search algorithms. Every row in the database contains all the information stored in the atoms object and in addition key–value pairs for storing extra quantities and for searching in the database.

Imagine a screening study looking for materials with a large density of states at the Fermi level. Storing the results in a database could then look like this:

```
from ase.db import connect
connection = connect('dos.db')
for atoms in ...:
    # Do calculation ...
    dos = get_dos_at_fermi_level(...)
    connection.write(atoms, dos=dos)
```

Here we have added a special `dos` column to our database, and we can now use the `dos.db` file for analysis with either the `ase.db` Python module (`connection.select('dos > 0.3')`) or the `ase db` command-line tool:



id ×	age ×	formula ×	energy ×	fmax ×	volume ×	magmom ×	↓ hform ×	dir_gap ×	ind_gap ×
486	7w	HfS_2	-20.435	0.007	265.030	-0.000	-1.588	2.075	1.311
404	7w	S_2Zr	-20.311	0.005	269.077	-0.000	-1.554	1.689	1.194
325	7w	HfS_2	-19.784	0.006	252.118	-0.000	-1.372	1.357	1.089
380	7w	S_2Zr	-19.742	0.009	256.082	0.000	-1.368	1.175	0.974
227	7w	Se_2Zr	-18.469	0.000	289.259	-0.000	-1.344	1.126	0.498
224	7w	$HfSe_2$	-18.517	0.010	285.748	0.000	-1.343	1.490	0.630
475	7w	S_2Ti	-18.758	0.002	230.894	0.000	-1.305	0.430	0.060
427	7w	S_2Sc	-16.565	0.000	276.111	-0.000	-1.226	0.000	0.000
398	7w	S_2Sc	-16.512	0.000	282.673	1.000	-1.209	0.000	0.000
407	7w	Se_2Zr	-18.036	0.012	277.710	0.000	-1.202	1.094	0.790

Figure 4. Showing the first 10 rows of the query `xc = PBE,O = 0` (`xc` key must be PBE and no oxygen atoms) sorted after formation energy, `hform`.

```
$ ase db dos.db "dos > 0.3"
```

The `ase db` tool can also start a local web server so that one can query the database using a web browser (see example in figure 4). By clicking on a row of the table shown in the web browser, one can see all the details for that row and also download the structure and data for that row. There are currently three database backends:

JSON Simple human-readable text file.

SQLite3 Self-contained, serverless, zero-configuration database. SQLite3 is built into the Python interpreter and the data is stored in a single file.

PostgreSQL Server-based database.

The JSON and SQLite3 backends work 'out of the box', whereas PostgreSQL requires a server.

4.2. Checkpointing

ASE includes a checkpointing module (`ase.calculators.checkpoint`) that adds powerful generic restart and rollback capabilities to scripts. It stores the current state of the simulation and its history using the ASE database discussed in section 4.1. This replaces the manual checking for previously written output files common to many scripts. The `Checkpoint` class takes care of storing and retrieving information from the database. This information always includes an Atoms object, and it can also include attached information on the internal state of the calculation. Code blocks can be wrapped into checkpointed regions using try/except statements, with the code block in the except statement being executed only if it was not completed in a previous run of the script. This allows one to quickly replay the script from cached snapshots up to the point where the script terminated in a previous run. The module also provides a `CheckpointCalculator` class which provides a shorthand for wrapping every single energy/force evaluation in a checkpointed region by wrapping the actual calculator so that calls to compute the potential energy or forces only carry out the calculation the first time the script is invoked. This is useful when each energy evaluation is slow (e.g. DFT), particularly when the overall runtime of the script is likely to exceed wall times typically available from the queueing system. Checkpointing capabilities therefore enable complex monolithic and reusable scripts whose execution spans a few or many runs on a high-performance computing system. In combination with a job management framework, this opens the possibility to encode and deploy robust automatic simulation workflows in ASE Python scripts, e.g. for combinatorial screening of material properties.

5. Calculators

The calculator constitutes a central object in ASE and allows one to calculate various physical quantities from an `Atoms` object. The properties that can be extracted from a given `Atoms` object depend crucially on the nature of the calculator

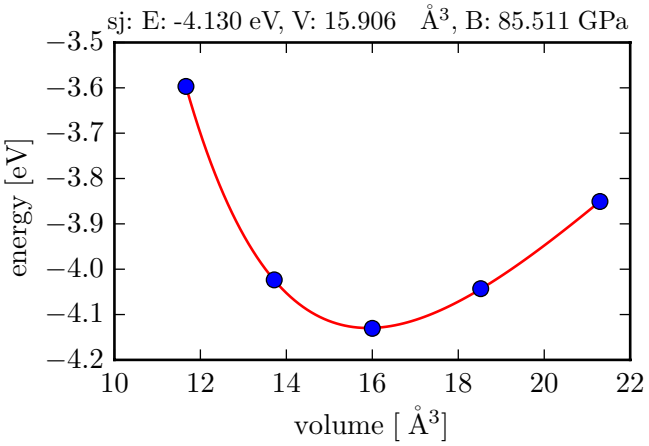


Figure 5. Potential energy as a function of volume of bulk Al. A four-parameter fit is applied to determine the optimal unit cell volume (V), energy (E) and bulk modulus (B).

attached to the atoms. For example, a DFT calculator may return properties such as the electronic density and Kohn–Sham eigenvalues, which are inaccessible with calculators based on classical interatomic potentials.

5.1. Energy and forces

An important method common to every ASE calculator is `get_potential_energy()`, which returns the potential energy of a given atomic configuration. In a quantum mechanical treatment of the electrons, this is the adiabatic ground state energy of the electronic system. Applying the method to two different atomic configurations will thus give the difference in energy between the two configurations.

A useful application of the method is illustrated by the equation of state module exemplified by the script below. The potential energy of fcc Al is calculated at various cell volumes and fitted using the stabilized jellium model [60]. The fit is shown in figure 5. This method provides a convenient way of obtaining lattice constants for solids.

```
from ase.eos import EquationOfState
from ase.build import bulk
from gpaw import GPAW

al = bulk('Al', 'fcc', a=4.0)
calc = GPAW(mode='pw', kpts=(4, 4 ,4))
al.calc = calc
cell = al.get_cell()

v = []
e = []
for x in [0.9, 0.95, 1.0, 1.05, 1.1]:
    al.set_cell(x * cell)
    v.append(al.get_volume())
    e.append(al.get_potential_energy())

eos = EquationOfState(v, e)
v0, e0, B = eos.fit()
eos.plot('eos_Al.pdf')
```

Another universal method carried by all calculators is `get_forces()`, which returns the forces on all atoms. The method is applied extensively when performing dynamics or structure optimization as described in section 6.

5.2. Communication schemes

While calculators are black boxes from a scripting perspective, there are some differences in how they interact with the environment. This section discusses how ASE deals with these technical aspects. The following section discusses parallelization, which is essential for most applications.

ASE calculators can be classified by their interactions with the underlying simulation code. At first, one can distinguish between calculators that run the simulation within the same Python interpreter process and those that launch a separate sub-process.

Representatives for the first class are e.g. the GPAW and EMT calculators. (More listed in table 1 and denoted as 'Python' for communication.) A big advantage of running the simulation within the same process is zero-copy communication. The calculator can simply pass instantiated ASE data-structures such as `atoms.positions` to the simulation code. In return, the simulation code can write its atomic forces directly into the buffer of a NumPy array. Another advantage is that it is easy for the calculator to store persistent information in memory that survives across many similar consecutive calculations, e.g. in a molecular dynamics or geometry optimization run. In those cases a significant speedup can be achieved by exploiting information from the previous steps. For example, a DFT code can reuse the previous wave-function as an initial guess for its next SCF optimization.

The second class of calculators execute the simulation code as a sub-process. (These are denoted as 'Interprocess' or 'Files' in table 1). This scheme is followed by the great majority of ASE calculators. Most of these calculators communicate with their sub-process through the filesystem. Hence, they generally perform the following four tasks:

1. The calculator generates an input file.
2. The calculator launches the simulation code as a sub-process.
3. The calculator waits for the sub-process to terminate.
4. The calculator parses the output files which were created by the simulation, and fills the ASE data-structures.

The main advantage of this scheme is its simplicity. It interacts with the simulation code in the same way as a user would. Hence, it does not require any changes to the simulation code itself. The big disadvantage of this scheme is the high I/O costs. When there are many consecutive invocations, a restart wave-function or electron density might have to be loaded from a file. If the simulation is MPI-parallelized, then the binary has to be accessed by each compute node before execution. Just creating the MPI session can already take several seconds [61].

Some file-based calculators like Quantum Espresso or Jacapo mitigate the start-up costs by keeping the simulation process alive across multiple invocations. The next calculation is triggered by writing a new input file, which the code automatically runs.

A way to avoid file I/O completely is to communicate via pipes. Such a scheme was recently implemented by the CP2K calculator [25, 62]. For this, the CP2K distribution comes with a small helper program called CP2K-shell. It provides a simple interactive command line interface with a well defined, parseable syntax. When invoked, the CP2K calculator calls `popen` [63] to launch the CP2K-shell as a sub-process and attaches pipes to its `stdin` and `stdout` file handles. This even works together with MPI, because the majority of MPI-implementations forward the `stdin`/`stdout` of the `mpiexec` process to the rank-0 process by default. The CP2K calculator also allows for multiple CP2K instances to be controlled from within the same Python process by instantiating multiple calculator objects simultaneously.

5.3. Parallel computing

Scientific computing is today usually done on computers with some kind of parallelism, either multiple CPU cores in a single computer, or clusters of computers often with multiple cores each. In the typical atomic-scale simulation performed with ASE, the performance bottleneck is almost always the calculation of forces and energies by the calculator. For this reason, ASE supports three different modes of calculator parallelization.

In the simplest case, a single process on a single core runs ASE, but whenever control is passed to the calculator, the calculation runs in parallel. This is the natural model whenever the interface to the calculator is file based: ASE writes the input files, starts the parallel calculation, and harvests the result.

Another model, for example used by the GPAW calculator, is to have ASE running on each CPU core with identical data. In this case Python itself is started in parallel e.g. by the mpiexec tool. This is only used with calculators having a native Python interface written for ASE. One has to be careful that all Python processes remain synchronized and with identical data. In this way, the data from ASE is already present in the Python process on all cores, and any necessary communication during the calculation is done by the calculator. Some care must be taken in the user's script when this model is used. First, data associated with the atoms must remain identical on all processes. This is particularly an issue if random numbers are used, for example in Monte Carlo simulations or Molecular Dynamics with the Langevin algorithm, where the random numbers must be generated either by a deterministic pseudorandom number generator, or on a single core and distributed to the rest. In most ASE modules using random numbers, this is done automatically. Second, care must be taken when writing output files. If more than one process writes to the same file, corruption is likely, in particular on network file systems. Printing from just one process may be dangerous, since asking the atoms for any quantity involving the calculator must be done on all processes to avoid a deadlock. ASE solves some of these issues transparently by providing helper functions such as `ase.parallel.paropen`. This function opens a file as normal on the master process, whereas data written by other processes is discarded. Since the ASE data is not distributed, this is sufficient for any normal output and does not require the user to think about parallelism.

For very large molecular dynamics simulations (millions of atoms), ASE is able to run in a distributed atoms mode. Currently, only the Asap calculator is able to run in this mode, and it needs to extend some modules normally provided by ASE. In this mode, the atoms are distributed among the processes according to their position in space (domain decomposition), each Python process thus only sees a fraction of the total number of atoms. If atoms move, they need to be transferred between processes; for performance reasons this is the responsibility of the calculator. When atoms thus migrate between processes, the number of atoms and their identities change in each Python process. Any module that stores data about the atoms internally, for example energy optimizers and molecular dynamics objects, will have their internal data invalidated when this happens. For that reason, Asap needs to provide specialized versions of such objects that delegate storage of internal data to the `Atoms` object. In the `Atoms` object, all data is stored in a single dictionary, and the calculator then migrates all data from this dictionary when atoms are transferred between processors.

When atomic configurations are written from a massively parallel molecular dynamics simulation, all information is normally gathered in the master process before being written to disk using one of ASE's supported file formats. In the most extreme simulations, gathering all data on the master process may be problematic (e.g. due to memory constraints). ASE supports a special file format for handling this case: the `BundleTrajectory`. The `BundleTrajectory` is not a file but a folder, with each atomic configuration in its own subfolder, and each quantity in one or more files. Normally, data would be written by a single process, and each quantity is written as an array into a single file, but in massively parallel simulations it is possible to have each process write its own data into separate files. ASE then assembles the arrays when the data is read again.

6. Dynamics and optimization

One is usually not only interested in static atomic structures, but also wants to study their movement under internal and external influences. ASE provides multiple algorithms for structure manipulation that can be used together with the calculator interfaces as was shown in the code example in section 2.1. The features supported by ASE and discussed in the following sections are: molecular dynamics with different thermodynamic controls, searching for local and global energy minima, or minimum-energy paths or transition states of chemical reactions. ASE further allows these types of structure manipulations to be restricted by complex constraints.

6.1. Molecular dynamics

The general idea of molecular dynamics (MD) is to numerically solve Newton's second law for all the atoms, thus generating a time-series from an initial configuration. The purpose of the molecular dynamics simulation may be to investigate the dynamics of a specific process, or to generate an ensemble of configurations in order to calculate a thermodynamic property. Many MD algorithms have been developed for related

but slightly different purposes (see e.g. [64]). This is reflected in the ASE code which supports a number of the more popular algorithms.

As Newton's second law preserves the total energy of an isolated system, so will any algorithm that integrates this equation of motion without modification: the simulation will produce a microcanonical (or *NVE*) ensemble with well-defined particle number, volume and total energy. One of the most popular such algorithms is velocity Verlet. In ASE this is implemented as a dynamics object:

```
from ase.units import fs
from ase.md.verlet import VelocityVerlet
dyn = VelocityVerlet(atoms,
                     timestep=5 * fs)
dyn.run(1000)  # Run 1000 time steps
```

A dynamics object shares many of the properties of an optimization object; it is possible, for example, to attach functions that are called at each time step, or after each N time steps. Useful objects to attach this way include Trajectories for storing the atomic configuration and `MDLogger`, which writes a log file with energies and temperatures.

6.1.1. Temperature control. Often, a constant-energy simulation is not what is desired, as the real system being modelled by the simulation is thermally coupled to its surroundings, and thus has a well-defined temperature. It is therefore necessary to be able to do simulations in the *NVT* ensemble without having to describe the coupling to the surroundings in details. ASE implements three different algorithms for constant-temperature MD: Berendsen dynamics, Langevin dynamics and Nosé–Hoover dynamics.

Berendsen dynamics [65] is conceptually the simplest: at each time step the velocities are rescaled such that the kinetic energy approaches the desired value over a characteristic time chosen by the user. While simple, this algorithm suffers from the problem that the magnitude of thermal fluctuations in the kinetic energy is not reproduced correctly, although this is mainly a problem for very small systems. Berendsen dynamics can be found in the `ase.md.nvtberendsen` module.

Langevin dynamics [66] adds a small friction and a fluctuating force to Newtons second law. While originally invented to simulate Brownian motion, it can be argued to be a physically reasonable approximation to the interactions with the electron gas in a metal. The main advantages of Langevin dynamics are that it is very stable and that the thermostat is *local*: if kinetic energy is produced in one part of the system, there is no risk that other parts cool down to compensate, as can be the case with other thermostats. A possible drawback is that Langevin dynamics is stochastic in nature, thus restarting a simulation will result in a slightly different trajectory. Langevin dynamics is implemented in the `ase.md.langevin` module.

Nosé–Hoover dynamics [67, 68] adds a single degree of freedom representing the heat bath; this degree of freedom is coupled to the velocities of the atoms through a rescaling factor. This method is very popular in the statistical mechanics community because it can be rigorously derived from a Hamiltonian. One major drawback of this method is that with only a single degree of freedom to describe the heat bath, oscillations may appear in this variable and thus in the temperature. While Nosé–Hoover dynamics is good at maintaining prescribed temperature, it is therefore less suitable to establish a specific temperature in the simulation. This problem can be addressed by introducing more auxiliary variables, the so-called Nosé–Hoover chain, but this is not implemented in ASE. Nosé–Hoover dynamics is implemented together with Parrinello–Rahman dynamics in the `ase.md.npt` module.

6.1.2. Pressure control. In addition to keeping temperature constant, it is often desirable to keep pressure (or the stress for solids) constant, leading to the isothermal-isobaric (*NpT*) ensemble. ASE provides two algorithms for *NpT* dynamics: Berendsen and Nosé–Hoover–Parrinello–Rahman.

Berendsen dynamics [65] allows for rescaling the simulation volume in addition to the kinetic energy, leading to the conceptually simplest implementation of NpT dynamics. This algorithm is implemented in the `ase.md.nptberendsen` module.

Nosé–Hoover–Parrinello–Rahman dynamics is a combination of Nosé–Hoover temperature control and Parrinello–Rahman pressure/stress control [69, 70]. ASE implements the algorithm set forth by Melchionna [71, 72]. As is the case for Nosé–Hoover dynamics, there is the possibility of oscillations in the auxiliary variables controlling both the temperature and the pressure, and the algorithm is more suitable for maintaining a given temperature and pressure than for approaching it. The ASE implementation allows for varying only the volume of the simulation box (suitable for constant-pressure simulations of e.g. liquids), and for varying both shape and volume of the box, possibly constraining the simulation box to remain orthogonal. In addition, constant strain rate simulations are possible where a dimension of the computational box is kept unaffected by the dynamics, but is assigned a constant rate of change. This is implemented in the `ase.md.npt` module.

6.2. Local structure optimizations

Local structure optimization algorithms start from an initial guess for the atomic positions and (mostly) use the forces acting on the atoms to find structures of lower energy in an iterative procedure until a given convergence criterion is reached. The methods available in ASE, in `ase.optimize`, are described below.

BFGS (Broyden–Fletcher–Goldfarb–Shanno) [18, 73]. This algorithm chooses each step from the current atomic forces and an approximation of the Hessian matrix, i.e. the matrix of second derivatives of the energy with respect to the atomic positions (see section 8.1). The Hessian is established from an initial guess which is gradually improved as more forces are evaluated.

L-BFGS is a low-memory version of the BFGS algorithm [18, 74, 75]. The full Hessian matrix has $\mathcal{O}(N^2)$ elements, making BFGS very expensive for force field calculations with millions of atoms. L-BFGS represents it implicitly as a series of up to n evaluated force vectors for a linear-scaling memory requirement of $\mathcal{O}(nN)$.

MDMin is an energy minimization algorithm based on a molecular dynamics simulation. From the initial positions, the atoms accelerate according to the forces acting on them. The algorithm monitors the scalar product $\mathbf{F} \cdot \mathbf{p}$ of the force and momentum vectors. Whenever it is negative, the atoms have started moving in an unfavourable direction, and the momentum is set to zero. The simulation continues with whatever energy remains in the system. An advantage of MDMin is that it is inspired by an intuitive physical process, but the algorithm does not converge quadratically like those based on Newton's method, and is therefore not efficient when close to the minimum.

FIRE (*fast inertial relaxation engine* [76]) likewise formulates an optimization through molecular dynamics. An artificial force term is added which 'steers' the atoms gradually towards the direction of steepest descent. FIRE uses a dynamic time step and other parameters to control the simulation. Again, if at some point the atoms would move against the forces, the velocities are set to zero and the dynamic parameters are reset. The FIRE algorithm often requires more iterations than BFGS as implemented in ASE, but the atoms move in smaller steps which can decrease the cost of a single self-consistent iteration.

SciOpt. ASE can use the optimization algorithms provided with SciPy for structure optimizations as well. However most of these general optimization algorithms are not as efficient as those designed specifically for atomistic problems.

Preconditioners can speed up optimization approaches by incorporating information about the local bonding topology into a redefined metric through a coordinate transformation. Preconditioners are problem dependent, but the general-purpose implementation in ASE provides a basis that can be adapted to achieve optimized performance for specific applications [77]. While the approach is general, the implementation is specific to a given optimizer: currently L-BFGS and FIRE can be preconditioned.

Tests with a variety of solid-state systems using both DFT and classical interatomic potentials driven though ASE calculators show speedup factors of up to an order of magnitude for preconditioned L-BFGS over standard L-BFGS, and the gain grows with system size. Precomputations are performed to automatically estimate all parameters required. A line search based on enforcing only the first Wolff condition (i.e. the Armijo sufficient descent condition) is also provided; this typically leads to a further speed up when used in conjunction with the preconditioner.

The preconditioned L-BFGS method implemented in ASE does not require external dependencies, but the `scipy.sparse` module can be used for efficient sparse linear algebra, and the `matscipy` package is used for fast computation of neighbour lists if available. `PyAMG` can be used to efficiently invert the preconditioner using an adaptive multigrid method.

6.3. Constraints

When performing optimization or molecular dynamics simulations one often wants to constrain the movement of the atoms. For example, it is common to fix the lower layers of a 'slab'-type adsorbate–surface model to the bulk lattice coordinates. This can be achieved by attaching the `FixAtoms` constraint to the atoms.

A number of built-in constraints are available in ASE. The user can easily combine these constraints or—if required—build their own constraints. The built-in constraints include:

- *Fix atoms.* Fixes the Cartesian positions of specified atoms.
- *Fix bond length.* Fixes a bond length between two atoms, while allowing the atoms to otherwise move freely.
- *Fixed-line, -plane, and -mode movement.* An atom can be constrained to move only along a specified line or within a specified plane; or, in fixed-mode, a system can be constrained to move along a specified mode only. An example of fixed-mode could be a vibrational mode.
- *Preserving molecular identity.* This constraint applies a restoring force if the distance between two atoms exceeds a certain threshold. In this way molecules can be prevented from dissociating. This class can also apply a restoring force to prevent an atom from moving too far away from a specified point or plane. The constraint was designed to work with techniques such as minima hopping in order to explore the potential energy surface while enforcing molecular identity [78].
- *Constraining internal coordinates.* Any number of bond lengths, angles, and dihedral angles can be constrained to fix the internal structure of a system.

An alternative to constraints is to use a 'filter', which works as a wrapper around the Atoms object when it is used with a dynamics method (optimization, molecular dynamics etc). In other words, the dynamics sees only the degrees of freedom that the filter provides and not all the atomic coordinates. The filter can thus hide certain degrees of freedom or make combinations of them. A number of filters are built into ASE, and again the user is free to build their own. The built-in methods include the following:

- *Masking atoms.* One can use a basic filter to fix the positions of specified atoms; this works by presenting *only* the positions, forces, momenta, etc, on the free atoms when the corresponding attributes are accessed. In particular for large-scale simulations, this can have the advantage of reducing the size of the Hessian matrix.
- *Optimizing unit cell vectors.* A filter can present the stresses on the unit cell along with the forces; this can be used to optimize the unit cell lattice vectors either simultaneously or independently from the atomic positions. These filters also present the strain of the unit cell as part of the positions attribute.

6.4. Transition states from the nudged elastic band method

Locating saddle points in a complex energy landscape is a common task in atomic simulations. For example, finding the saddle point is required to determine the energy barrier for diffusion of an adsorbate across a surface for a chemical reaction

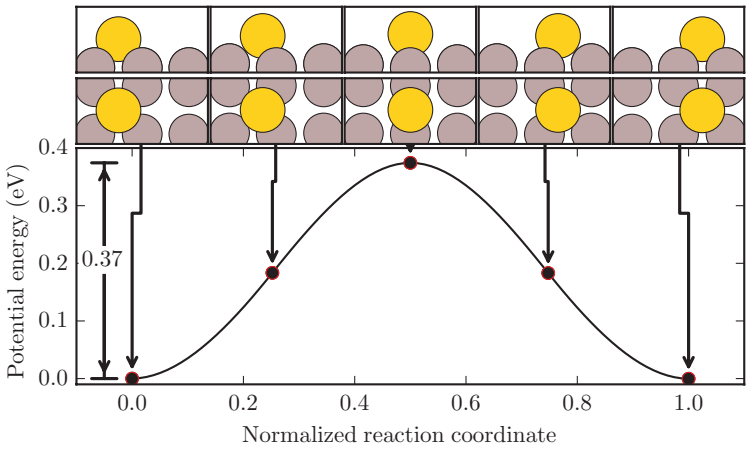


Figure 6. The potential energy as a function of the normalized reaction path for the diffusion path of a Au atom on the Al(001) surface.

event (bond breaking or formation). To locate saddle points within the harmonic approximation, ASE offers the *nudged elastic band* method and the *dimer* method.

The nudged elastic band (NEB) method [9, 10] as formulated by Henkelman and Jónsson [79, 80] works by creating a series of Atoms objects between two local minima. These Atoms objects, *images*, are then relaxed in order to determine the lowest-energy pathway. So-called 'springs' are inserted between adjacent images in order to apply a restoring force on each image which prevents them from relaxing into each other and the starting or ending minima. At the same time the component of the force from the energy landscape that is parallel to the band is removed resulting in the nudged elastic band. The force then have two components; one from the energy landscape perpendicular to the band converging the band towards a minimum energy path (MEP) and the spring force that secures the images are equally spaced.

The NEB method is accessed by importing `NEB` from the `ase.neb` module. `NEB` accepts as input a series of initial images with attached calculators. The initial images can be acquired e.g. from interpolation of an initial and a final state between which the pathway is desired, or from a previous pathway relaxed with another energy descriptor. After the NEB object is created, it is handed to the chosen optimizer and the relaxation of the pathway is initialized. The end result is a series of images describing the lowest-energy pathway.

In ASE, the NEB method is implemented on top of the a normal relaxation scheme. For each image, the assigned optimizer determines the forces on each atom, and these calculated forces are then modified by the NEB method. Thus, before the atoms are moved, the restoring forces are applied between each image to maintain the pathway.

The following is an example of a gold atom diffusing on top of an aluminium (001) surface. The upper panel of figure 6 shows the atom configuration. First, the script initializes the initial and final images of the gold atom placed into two neighbouring hollow sites. It then relaxes these two images, which will serve as end-points for the NEB path. Next, the intermediate images are initialized so that they linearly interpolate the initial and the final state. This is done by creating several copies of the atoms and passing them to `neb.interpolate()`.

```
from ase.calculators.emt import EMT
from ase.neb import NEB
from ase.optimize import BFGS
from ase.io import write
from ase.build import fcc100, add_adsorbate
from ase.constraints import FixAtoms

# 2x2-Al(001) surface with 3 layers and an
# Au atom adsorbed in a hollow site:
initial = fcc100('Al', size=(2, 2, 3))
add_adsorbate(initial, 'Au', 1.7, 'hollow')
initial.center(axis=2, vacuum=4.0)

# Fix second and third layers:
mask = [atom.tag > 1 for atom in initial]
initial.set_constraint(FixAtoms(mask=mask))

# Initial state:
initial.calc = EMT()
qn = BFGS(initial)
qn.run(fmax=0.05)

# Final state:
final = initial.copy()
final[-1].x += final.get_cell()[0, 0] / 2
final.calc = EMT()
qn = BFGS(final)
qn.run(fmax=0.05)

images = [initial]
for i in range(3):
    image = initial.copy()
    image.calc = EMT()
    images.append(image)
images.append(final)
```

```
# NEB object with new interpolated images
neb = NEB(images)
neb.interpolate()
qn = BFGS(neb, trajectory='neb.traj')
qn.run(fmax=0.05)

write('output.traj', images)
```

The above script produces an output file containing the relaxed pathway used to produce figure 6.

Finding the true saddle point for more complex pathways is not trivial. The best initial guess for the reaction path may not be a linear interpolation between initial and final image, but instead be related by rotations of subgroups of atoms. An improved preliminary guess for the minimum energy path can be generated using images on the *image dependent pair potential* (IDPP) surface [81]. Optimization of all atom pair distances toward an interpolation of all atom pair distances for all intermediate images along the path results in an initial path much closer to the MEP.

Once a good approximate reaction pathway has been determined, the *climbing-image* extension of the NEB module can be invoked to converge the highest-energy image to the nearest saddle point. The method works by omitting spring forces on the highest energy image and inverting the force it experiences parallel to the path. The climbing image is still allowed to relax down the energy landscape perpendicularly to the lowest-energy path like all other intermediate images. Because the additional freedom of the climbing image makes this calculation computationally more expensive, it is advised that this is only done when a good guess of the saddle point is available.

Additional NEB extensions are available in the module. For instance, the full spring force is omitted by default and only the spring force parallel to the local tangent is used together with the true force (as evaluated by the calculator) perpendicular to the local tangent. A full list of capabilities is available on the ASE website.

The standard NEB algorithm distributes the assigned computational resources equally to all the images along the designated path. This implementation results in an inflexible and potentially inefficient allocation of resources, given that each image has a different level of importance towards finding the saddle point. A dynamic resource allocation approach is possible through the *AutoNEB* [82] method in `ase.autoneb`. AutoNEB uses a simple strategy to add images dynamically during the optimization.

AutoNEB first converges a rough reaction path with a few images using standard NEB. Once converged, an image is added either where the gap between the current images is largest, or where the energy slope is greatest. The reaction path is refined by iteratively adding images and reconverging the pathway.

The virtue of the AutoNEB method is that it is possible to define a total number of internal images which is greater than the number of images to simultaneously participate in the optimisation. Some images will then be moving while others are frozen. Whenever an image is added, the moving images will be those closest to the most recently added one. This feature allows for computational resources to always be focused on the most important region of the pathway. The method has been utilized for a number of examples [83–85] with benchmarking cases with 50–70% reduced computational cost relative to the standard algorithm [85].

For systems with no fixed atom positions and/or periodic boundary conditions, overall rotation and translation are external changes with no internal structural changes. For normal NEB calculations involving nanoparticles, external structural changes can pose problems for the convergence to the minimum energy path. The system will seek to avoid high energy areas and hence rotate and/or translate away from these, which is not possible to the same extent for a constrained system. The NEB module supports the NEB-TR [86] method, which speeds up convergence for such systems by removing rotation and translation movement.

6.5. Transition states from the dimer method

Like the NEB method above, the dimer method is used to find saddle points in multidimensional potential energy landscapes. Unlike NEB, the dimer method only requires a starting basin and is useful for finding most or all relevant pathways out of that basin.

The dimer method is a minimum mode following method which estimates the eigenmode corresponding to the lowest eigenvalue of the Hessian (minimum mode) and traverses the potential energy surface (PES) uphill in that direction. Eventually, it reaches a first order saddle point with exactly one negative eigenvalue.

The dimer method can be split into three independent phases.

1. Estimating the minimum mode.
2. Inverting the gradient along the minimum mode to make first order saddle points appear as minima.
3. Move the system according to the inverted gradient.

Only the first of these phases is unique to the dimer algorithm. Other methods estimate the minimum mode differently. The dimer method is implemented in ASE in such a way that it should be straightforward to include other minimum mode estimators.

To find a saddle point, the system is initially located at an energy minimum and randomly displaced [87]. The displacement achieves two things: first, it moves the system away from a zero gradient location (the minimum), and secondly, it can be used as the seed to sample as many saddle points as possible.

The dimer method identifies the minimum mode by making an estimate of the curvature of the PES along a given unit vector, $\hat{\mathbf{s}}$, and then iteratively rotates $\hat{\mathbf{s}}$ until it reaches an energy minimum. This energy minimum represents the lowest curvature.

The name of the dimer method is derived from the way that $\hat{\mathbf{s}}$ is defined. Two *images* are chosen: one with the input system coordinates and the other displaced along $\hat{\mathbf{s}}$ by a distance of Δ_D. The gradients at each image are then used to estimate the 2nd derivative of the PES using finite difference. The force components perpendicular to $\hat{\mathbf{s}}$ are used to determine the torque which rotates the dimer to obtain

lower energy, iteratively reaching the estimate of the minimum mode.

The dimer method is implemented in ASE through the `DimerAtoms` class, which extends the `Atoms` class with information relevant to the dimer method, such as the minimum mode estimate and the curvature. The system can initially be displaced from the energy minimum configuration by a predefined vector, by selecting certain atoms to be randomly displaced or by defining a sphere in which all atoms are randomly displaced.

A default dimer calculation can be set up by creating a `DimerAtoms` object from an `Atoms` object with a calculator attached. The parameters controlling the calculation can either be passed directly to the `DimerAtoms` object when created or can be specified using a `DimerControl` object.

Multiple control parameters are defined in the `DimerControl` but the most important ones have to do with Δ_D and the amount of rotations allowed before performing an optimization step:

- Rotational force thresholds to define the conditions under which no more rotations will be performed before performing an optimization step.
- The maximum number of rotations to be performed before making an optimization step.
- Δ_D, the separation of the dimer's images. In order for an accurate finite difference estimate of the 2nd derivative, this should be kept small, but still large enough to avoid effects of noise in the force calculations.

7. Global optimization

Finding the atomic configuration with the lowest possible energy is much more challenging than finding just a local minimum. The number of local minima grows at least exponentially with the number of atoms (the existence of about 10^6 local minima for a 147-atom Lennard-Jones cluster has been suggested [88]), and finding the global minimum is therefore a daunting task. One of the main challenges for global optimization is that different local minima might be separated by high energy barriers which much be overcome in order to move between local minima.

One common approach to this problem is simulated annealing, in which the atomic system is initially equilibrated at a certain temperature using, for example, molecular dynamics. After this, the temperature is decreased slowly to identify low energy configurations. However, this method is not always efficient and a number of alternative global optimization techniques have been developed.

ASE provides three methods for global optimization: basin hopping, minima hopping and genetic algorithms.

7.1. Basin hopping

In the basin hopping method [88], the atoms perform a random walk. Every structure visited is relaxed with a local optimization method, and the unrelaxed structure is then assigned the energy of the obtained minimum. Thus all structures within the same *basin* are considered to have the same energy. In this way the barriers between close-lying local minima are effectively removed, and the PES becomes a step-function with the energies defined by the local minima (see the illustrative figure 2 in [88]). The modified PES is then explored by Monte Carlo at an adjustable temperature: going from one basin to another is accepted at random depending on how favourable its energy is.

7.2. Minima hopping

A more automated approach is minima hopping [89], in which one uses alternating MD and local minimization steps in combination with self-adjusting parameters to explore the potential energy surface. Briefly, a MD step at a specified initial temperature is used to randomly 'shoot' the structure out of a local minimum region in the PES; after the MD trajectory encounters a specified number of path minima, the structure is optimized to the next local minimum. If the minimum found is identical to any previously-found minimum, the MD temperature is increased and the step is repeated. Otherwise the algorithm first lowers the search temperature and then decides to accept the step if the new local minimum is no higher than a specified energy difference above the previous step. If the found local minimum is accepted, it is added to a list of found minima and the energy difference threshold is decreased. If it is rejected, the energy difference threshold is increased. In this way, a list of local minima is generated while two parameters—the search temperature and the acceptance criterion—are automatically adjusted in response to the success of the algorithm. The ASE implementation of minima hopping allows the user to easily customize the key features of the algorithm, such as the local optimizer employed or the molecular dynamics scheme. It is also possible to perform parallel searches which share the list of found minima. The minima hopping scheme can also be combined with constraints, for example to prevent molecules from dissociating [78].

7.3. Genetic algorithms

In addition to the global optimization schemes described in sections 7.1 and 7.2, ASE also contains `ase.ga`, a genetic algorithm [90–93] (GA) module. A GA takes a Darwinistic approach to optimization by maintaining a *population* of solution *candidates* to a problem (e.g. what is the most stable four component alloy [94]?). The population is evolved to obtain better solutions by mating and mutating selected candidates and putting the fittest *offspring* in the population. The *fitness* of a candidate is a function which, for example, measures the stability or performance of a candidate. Natural selection is used to keep a constant population size by removing the least fit candidates. Mating or *crossover* combine candidates to create offspring with parts from more candidates present, when favorable traits are combined and passed on the population is evolved. Only performing crossover operations risks stagnating the evolution due to a lack of diversity—performing crossover on very similar candidates is unlikely to create progress

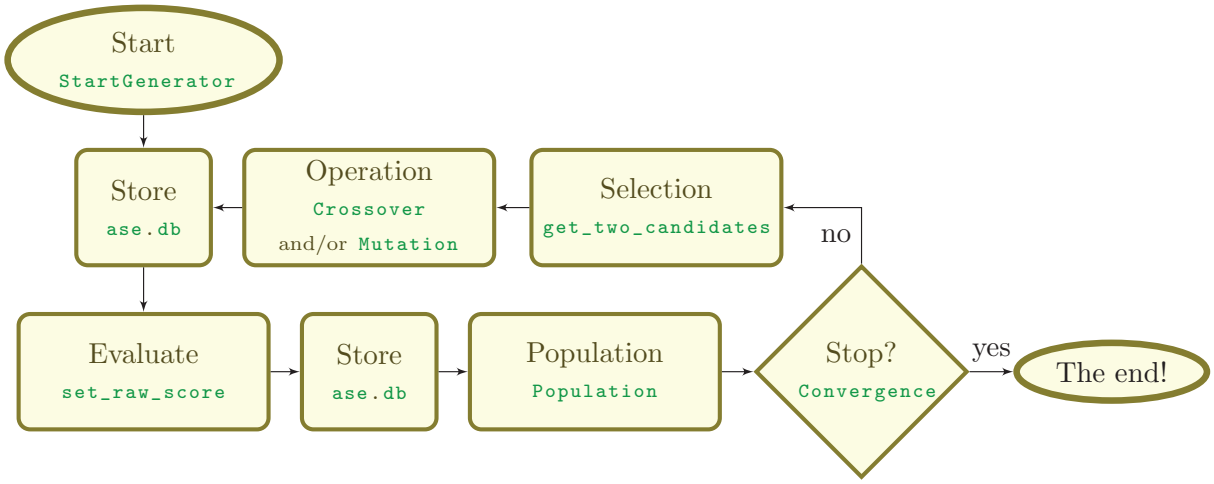


Figure 7. Flow of the genetic algorithm. Examples of functions or classes at each step are shown in green.

when performed repeatedly. *Mutation* induces diversity in the population and thus prevents premature convergence.

GAs are generally applicable to areas where traditional optimization by derivative methods are unsuited and a brute force approach is computationally infeasible. Furthermore, the output of a GA run will be a selection of optimized candidates, which often will be preferred over only getting the best candidate, especially taking into account the potential inaccuracy of the employed methods. Thus a GA finds many applications within atomic simulations, and will often be one of the best methods for searching large phase spaces. We will present a couple of usage cases in section 7.3.1.

7.3.1. Usage examples of the GA implementation in ASE. The `ase.ga` implementation has been used to determine the most stable atomic distribution in alloy nanoparticle catalysts [95, 96]. For this purpose, specific permutation operators were implemented promoting the search for different possible atomic distributions within particles, i.e. core/shell, mixed or segregated. For example the operator promoting core/shell particles permutes two atoms in the central and surface region respectively, a mixed (segregated) particle is promoted by permuting two atoms each in local environments with a high (low) density of identical atoms. These operators, if used dynamically, greatly improved the convergence speed of the algorithm.

The most stable Au_{6-12} clusters on a TiO_2 support were also determined using `ase.ga` [97]. The approach, inspired by Deaven and Ho [98], implemented the cut-and-splice operator and utilized the flexibility of ASE to perform local optimization with increasing levels of theory. This led to the discovery of a new global minimum structure. The GA implementation was benchmarked for small clusters and described in greater detail in [99].

Bulk systems are also readily treated in `ase.ga`; for example, `ase.ga` was used in a search for ammonia storage materials with high storage capacities and optimal ammonia release characteristics. The best candidates, some with a record high accessible hydrogen capacity, were subsequently experimentally verified [100, 101]. Operators utilizing chemical trends were implemented, e.g. mutating an atom to a neighboring element in the periodic table (up, down, right or left). This approach is readily transferable to other screening studies where the phase space is too great for a full computational screening.

7.3.2. GA implementation in ASE. The implementation requires some insight from the user on how the algorithm should treat the problem, e.g. which operators are appropriate or when is convergence achieved. The only way to get the algorithm running optimally is by testing the use of different functions and parameters for the given search objective. However, even without optimization of parameters, the GA can perform reasonably well. In the following we shall discuss the different components in a GA calculation in ASE (also shown in the flow chart figure 7), and provide an example for optimization of a small cluster on a gold surface. Other GA examples are available on the ASE web-pages.

Start The initial population must be generated to start. A helper function, `StartGenerator`, is supplied for atomic clusters. The initial population can be random (unbiased) or comprise seeded suggestions, e.g. candidates the user thinks will be good (biased). The subsequent steps are then repeated until some convergence criterion is reached.
Store Unevaluated candidates (initial or offspring) are saved in the database.
Evaluate Candidates are typically evaluated in a separate script, allowing the search to progress in parallel through a queueing system. The evaluation will typically involve a local geometry optimization. The main objective of the evaluation script is to calculate the *raw score*, i.e. the fitness of the candidate without taking the rest of the population into account. Evaluated candidates and their raw scores are added to the database.
Population The population step determines how to calculate the fitness from the raw score. It is useful to compare candidates collectively and only keep the fittest one of similar candidates. This can also be achieved by penalizing the fitness of similar candidates.
Stop The GA run is normally considered converged when no evolution of the fittest candidates takes place for a cer-

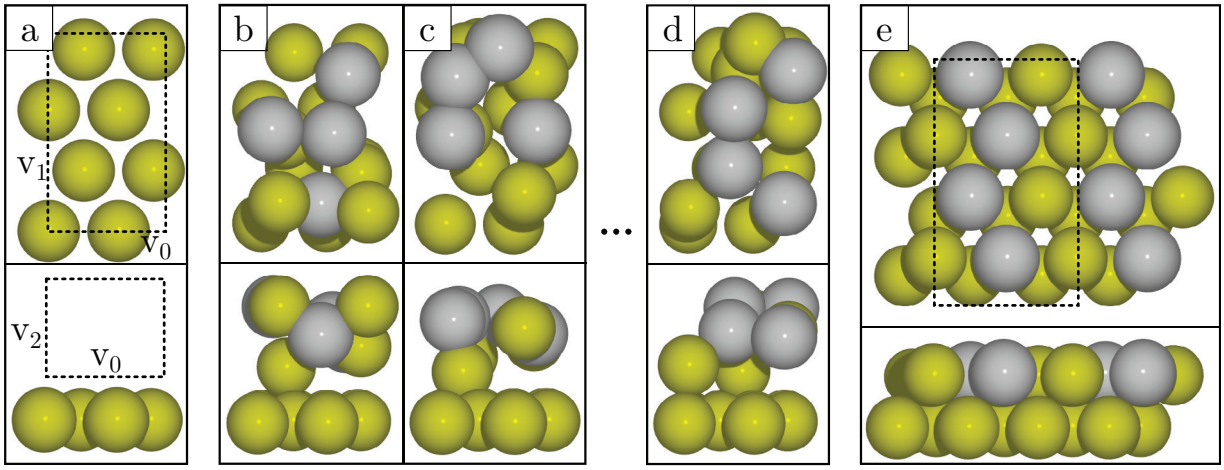


Figure 8. (a) The slab with the predefined box where the initial atoms are shuffled. (b)–(d) Three random examples of candidates created by the start generator. (e) The overall surface structure for the global minima with the EMT potential. The figures above are the top view and the bottom figures are the side view of the system.

tain amount of time. It is also possible to set a hard limit for the number of candidates that should be generated and conclude the search once this criterion has been met.

Selection This step performs the selection among the current population, e.g. by the roulette wheel method. This can function in two ways:

1. Generational where the current population is used as parents to form a new generation of candidate offspring. The new generation and current population are then combined to form the new population. However, since the algorithm halts until the evaluation of a full generation is finished, this does not parallelize optimally.
2. The population can be treated as a pool from which candidates are used as parents and to which new candidates are added continually. This keeps the computational load constant.

Operation The types of operators to use in a search are determined by the specific problem. It is always desirable though to use both *crossover* and *mutation*, however not necessarily in equal amounts—it is possible to give a certain weight to any operator, that can be changed dynamically [95]. It should be noted that, for a problem well suited to a GA, it will be beneficial to set a greater likelihood of crossover compared to mutation. The newly created offspring is added to the database and the whole cycle repeats.

If the user has a new idea, the easiest way to implement it is to modify a copy of one of the existing classes.

Once the GA script runs, no action is required from the user, but it is beneficial to check the evolution by investigating the db file as described in section 4.1. It is typical to run a GA several times with identical or slightly different initial conditions to verify the global minimum. If allowed by the nature of the problem, it is common to utilize a central database to check during the evaluation step if the candidate has been calculated in a previous run or in another instance of the GA running in parallel.

7.3.3. Running a genetic algorithm. In the following we outline a complete GA search for maximally stable configurations of a gold/silver overlayer on top of a $2 \times 4 \times 1$ Au(1 1 1) slab The complete code for the example can be found on the ASE web page.

First, the gold slab is created. A random initial population is then produced on top of the slab (as displayed in figure 8(a)) and added to the database using the two `ase.ga` objects `StartGenerator` and `PrepareDB`. The start population is subsequently relaxed before the GA run is initiated. Examples from an unrelaxed start population can be found in figures 8(b)–(d).

The next step is to set up the population, pairing scheme, and types of mutation before the population is evolved. This is facilitated by helper classes for cut-and-splice pairing and permutation mutations. In this case, 50 new candidates are initialized either by mutation (30% permutation mutation in this case) or by a crossover pairing of two candidates from the current population. New candidates are relaxed in turn while keeping the population updated with the fittest 20 unique candidates after each new child is added to the database. Setting up the population and running the main loop can be done as shown in the code example below.

```
# Create the population
n_to_optimize = len(atom_numbers)
comp = InteratomicDistanceComparator(
    n_top=n_to_optimize)
population = Population(
    data_connection=data,
    population_size=20, comparator=comp)

pair = CutAndSplicePairing(slab,
    n_to_optimize, blmin).get_new_individual
mutate = PermutationMutation(
    n_to_optimize).get_new_individual

# Test 50 new candidates
for i in range(50):
```

```
    # Parents a and b:
    a, b = population.get_two_candidates()
    # Mutate (30 %) or pair parents a
    # and b to obtain child candidate:
    if random() < 0.3:
        child, desc = mutate([a])
    else:
        child, desc = pair([a, b])
    if child is None:
        continue
    data.add_unrelaxed_candidate(child,
        description=desc)

    relax_and_add(child, data)
    population.update()

write('all_candidates.traj',
      data.get_all_relaxed_candidates())
```

In the end all tested candidates are automatically listed by their raw score (potential energy in this example) and written to a trajectory file. The global minimum structure for this specific problem can be seen in figure 8(e).

8. Vibrations and thermodynamics

The vibrational characteristics of a molecule or system of interacting atoms describe the dynamics near a stable geometry within the harmonic oscillator approximation. Vibrational analysis provides insight into local dynamics while remaining relatively inexpensive compared to fully integrating the equations of motion from an initial state, e.g. using molecular dynamics with `ase.md`. In molecules, the key quantity of interest is the set of vibrational frequencies and their corresponding normal modes. Similarly, vibrations in periodic crystals are described by a continuum of vibrational states, known as the phonon band structure. Vibrational analysis provides a direct link to experimental spectroscopic values. Moreover, vibrational frequencies are necessary for the calculation of thermodynamic functions in `ase.thermochemistry` within the ideal gas and harmonic approximations.

8.1. Molecular vibrations

We begin by expanding the potential energy function with respect to atomic positions, u_i:

$$E = E_0 + \sum_i^{3N} \left.\frac{\partial E}{\partial u_i}\right|_0 (u_i - u_{i0}) + \frac{1}{2}\sum_i^{3N}\sum_j^{3N} \left.\frac{\partial^2 E}{\partial u_i \partial u_j}\right|_0 (u_i - u_{i0})(u_j - u_{j0}) + \cdots \quad (1)$$

Here, indices i and j run over the three coordinate axes of N atoms and the subscript 0 indicates the reference geometry. The first term in the expansion accounts for an arbitrary shift of the total potential energy, so we are free to set it equal to zero. Assuming the expansion is about a stationary point on the potential energy surface (usually energy minima and occasionally saddle points are of interest), the derivative in the linear term of the expansion is zero. Ignoring higher-order terms, we may rewrite the expansion as

$$E = \frac{1}{2}\sum_{i,j} \Delta u_i H_{ij} \Delta u_j = \frac{1}{2}\mathbf{\Delta u}^{\mathrm{T}}\mathbf{H}\mathbf{\Delta u}. \quad (2)$$

$\mathbf{H}$ is called the Hessian matrix and is defined as

$$H_{ij} = \left.\frac{\partial^2 E}{\partial u_i \partial u_j}\right|_0 = -\frac{\partial F_j}{\partial u_i}. \quad (3)$$

In the above expression, F_j denotes the force along the atomic coordinate axis j. $\mathbf{H}$ is constructed from finite-difference first derivatives of the forces obtained from an attached ASE calculator. Each atom is displaced forwards and backwards along each coordinate axis, requiring $6N + 1$ total force evaluations. Note that, by construction, $\mathbf{H}$ is symmetric. It follows that the eigenvalues of $\mathbf{H}$ are real and its eigenvectors form a complete orthogonal basis. Writing Newton's equations of motion in terms of the Hessian yields

$$-\mathbf{H}\mathbf{\Delta u} = \mathbf{M}\frac{\mathrm{d}^2\mathbf{\Delta u}}{\mathrm{d}t^2}, \quad (4)$$

which is solved by $\mathbf{u}_k(t) = \mathbf{a}_k \exp(-\mathrm{i}\omega_k t)$. Plugging in this solution, the equation of motion becomes

$$\mathbf{H}\mathbf{u}_k = \omega_k^2 \mathbf{M}\mathbf{u}_k, \quad (5)$$

where $\mathbf{M}$ is a diagonal matrix of the atomic masses. This is then solved as a standard eigenvalue problem. The eigenvalues of the mass-weighted Hessian $\mathbf{M}^{-1/2}\mathbf{H}\mathbf{M}^{-1/2}$ are the squared vibrational frequencies and the eigenvectors are the mass-weighted normal modes $\mathbf{M}^{1/2}\mathbf{u}_k$.

Many calculators represent quantities such as the electron density on a real-space grid. This gives rise to the so-called *egg-box* effect: The forces on each atom vary artificially under translational movement of the grid relative to the atom, leading to inaccuracies in the Hessian. This dilemma may be resolved for isolated systems by imposing momentum conservation [102] such that $\sum_i H_{ij} = 0$ for all j. In ASE, this condition is enforced by adjusting the diagonal elements through $H_{jj} = -\sum_{i \neq j} H_{ij}$, which preserves the symmetry of $\mathbf{H}$.

The effect is to replace the force on an atom when displaced from its equilibrium position with *minus* the sum of the forces on all *other* atoms under the same displacement, averaging the egg-box noise over the whole system.

In practice, the vibrations of a molecule described by an `Atoms` object is found by creating a `Vibrations` object and running it:

```
from ase.vibrations import Vibrations
vib = Vibrations(atoms)
vib.run()
```

The `Vibrations` object can then be queried for information about the calculated vibrational modes.

8.2. Spectral properties

Post-processing methods are available in `ase.vibrations` for calculating spectral properties related to vibrations. These methods use quantities that are calculated via finite difference displacements. Infrared spectra can be obtained from changes in molecular dipoles [103] that are available from electronic structure calculators.

Raman spectra are more involved, as they require excited state properties that are not as commonly provided by calculators as ground state properties. The ASE modules are intended to be usable with different caculators, but were tested so far with GPAW only. These modules are a work in progress, but special topics like the evaluation of approximate Raman spectra [104] and the calculation of Franck–Condon overlaps depending on Huang–Rhys factors derived from excited state forces already exist.

8.3. Phonons

Treatment of lattice dynamics in terms of phonons is essential for describing many properties of crystalline solids including thermodynamic functions (see `ase.thermochemistry`), superconductivity, infrared and Raman absorption, phase transitions and sound propagation [105]. The `ase.phonons` module is used to calculate phonon frequencies, which can be used to construct band structures and densities of states and visualize the time-dependent motion of the phonon modes.

The calculation of phonon modes may be thought of as an extension of the vibrational analysis described in section 8.1 to a periodic lattice. The phonon modes are characterized by a wave vector **k**, which specifies the direction and spatial frequency of propagation.

Two well-established *ab initio* methodologies for performing phonon analyses are the finite displacement method, also called the direct method [106], and the linear response method [107]. `ase.phonons` uses the former, in which the primitive cell is repeated across its periodic boundary conditions to form a supercell. Ideally, the supercell should be large enough that no atom interacts with any of its periodic images in neighboring cells. In ordinary metals, interatomic forces decay rapidly. In polar materials, the dipole–dipole forces are longer in range. For reliable results one should therefore always check any phonon-related observable for convergence with respect to supercell size.

The matrix of force constants, **H**, is calculated by displacing the atom or atoms in one primitive cell within the supercell along each Cartesian axis and calculating the force response on all other atoms in the supercell. It is only necessary to perform the displacement on the atoms in this one *dynamic* primitive cell because all other primitive cells in the crystal are related by translation of a lattice vector. Thus, constructing **H** requires $6N + 1$ force evaluations on the entire supercell, where N denotes the number of atoms in the *dynamic* primitive cell. A dynamical matrix **D** may then be defined in terms of **H** as

$$D_{jj'}(\mathbf{k}) = \frac{1}{\sqrt{m_j m_{j'}}} \sum_{n' \in \text{supercell}} H_{j0,j'n'} \exp(i\mathbf{k} \cdot (\mathbf{r}_{j0} - \mathbf{r}_{j'n'})), \quad (6)$$

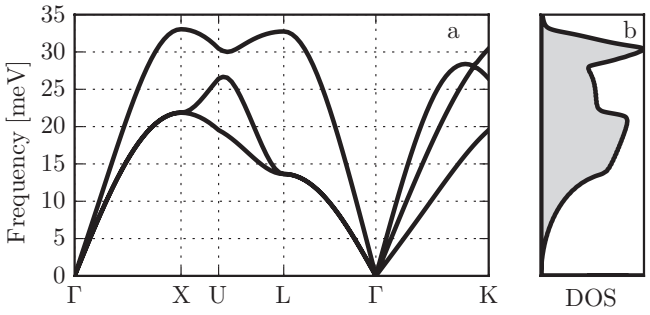


Figure 9. Continuum of vibrational states for bulk FCC aluminum. (a) Phonon band structure calculated with a $7 \times 7 \times 7$ supercell using effective medium theory, i.e. the EMT calculator in ASE. (b) The phonon density of states for the same system, integrated over an equispaced $50 \times 50 \times 50$ grid.

where n' labels the unit cells and the subscript zero indicates the primitive cell where atoms are displaced. Solving Newton's equations of motion requires solving for the eigenvalues of the dynamical matrix.

$$\mathbf{D}(\mathbf{k})\mathbf{u}_\mathbf{k} = \omega_\mathbf{k}^2 \mathbf{u}_\mathbf{k} \quad (7)$$

As in (5), the eigenvalues of the dynamical matrix are the squared frequencies. In this case, each frequency corresponds to a phonon mode $\mathbf{u}_\mathbf{k}$. Note that the dynamical matrix is a function of the wave vector **k**. Consequently, each wave vector is associated with $3N$ frequencies as can be visualized on a band structure plot (figure 9). Unlike molecular vibrations, phonon frequencies are dispersed along a continuous density of states, which is constructed by sampling the frequencies at many k-points. This density of states is necessary for the calculation of thermodynamic functions of crystals in `ase.thermochemistry`. The egg-box effect is minimized by enforcing momentum conservation such that

$$\sum_{j,n'} H_{j0,j'n'} = 0, \qquad j' \in \{1 \ldots 3N\}. \quad (8)$$

Note that this is a stricter translational invariance condition than that imposed by for molecules, and does not preserve the symmetry of **H**. The condition must be applied to **H** and followed by symmetrization in an iterative process until convergence is attained.

In practice, a phonon calculation is performed much like a calculation of molecular vibrations, except that often the number of unit cells used are specified:

```
from ase.phonons import Phonons
ph = Phonons(atoms, calc,
             supercell=(7, 7, 7))
ph.run()
```

The `Phonons` object can then be queried for information about the different phonon modes.

8.4. Thermochemistry

A common task in atomistic simulations is to convert the output of electronic structure calculations, which represent

calculations of single configurations on the potential energy surface (PES), into thermodynamic properties, which are statistical functions of the energetics over an ensemble of states. For example, optimization routines within ASE can find a local minimum on the PES; however, to convert this potential energy E_{pot} into a thermodynamic quantity such as the free energy, a statistical-mechanics treatment that utilizes the shape of the PES near the local minimum is necessary, along with an appropriate model. Three standard cases are implemented in ASE (within `ase.thermochemistry`) to convert PES information into thermodynamic quantities (ignoring electronic excitation contributions to the heat capacity); given the modular nature of ASE, the user can readily expand upon these basic methods with customized methods as needed, for example to deal with anharmonicity or electronic contributions. The three standard approaches are listed below.

- *The harmonic limit.* In the simplest case, all $3N$ atomic degrees of freedom (DOFs) are treated as separable and harmonic within the accessible temperature range; the vibrational energy levels ϵ_i corresponding to these motions can be produced from a normal-mode analysis as described in section 8.1. This is the same treatment as in the ideal-gas limit, described below, but without translational or vibrational DOFs. A common example of this limit is the examination of the thermodynamics of an adsorbed species on a catalytic surface [108]; the $3N$ DOFs of the adsorbate are then assumed to be harmonic and independent of the modes in the surface. This allows the calculation of properties such as internal energy U and entropy S at a specified temperature T with the Helmholtz energy $F = U - TS$. These methods are available via the `HarmonicThermo` class. The internal energy is taken to be

$$U(T) = E_{\mathrm{pot}} + \Delta E_{\mathrm{ZPE}} + \sum_{i}^{3N} \frac{\epsilon_i}{\mathrm{e}^{\epsilon_i/k_{\mathrm{B}}T} - 1}, \tag{9}$$

  where the zero-point energy ΔE_{ZPE} has its usual definition of $\frac{1}{2}\sum_i \epsilon_i$ and k_{B} is the Boltzmann constant. The entropy is found from

$$S(T) = k_{\mathrm{B}} \sum_{i}^{3N} \left[\frac{\epsilon_i}{k_{\mathrm{B}}T(\mathrm{e}^{\epsilon_i/k_{\mathrm{B}}T} - 1)} - \ln(1 - \mathrm{e}^{-\epsilon_i/k_{\mathrm{B}}T}) \right]. \tag{10}$$

- *The ideal-gas limit.* In the limit of an ideal gas, one assumes that the $3N$ atomic DOFs can be treated independently and separated into translational, rotational, and vibrational components [109, 110]. Three-dimensional gases have three translational DOFs. General polyatomic gases have an additional three rotational DOFs; although linear molecules have only two rotational DOFs and monotonic gases have none. The remaining DOFs are vibrations which are treated harmonically in this limit. This allows for the calculation of such properties as the enthalpy H, entropy S, and Gibbs free energy G. Aside from ideal gases, this method can be used to estimate the properties of condensed species via thermodynamic arguments. For example, the direct calculation of the free energy of liquid water at 298 K from first principles is daunting. However, one can calculate the free energy of water vapor in equilibrium with liquid water (which, at a vapor pressure of 3.2 kPa is well described by the ideal-gas approximation), and equate this to the free energy of liquid water via the thermodynamic criterion for equilibrium. These methods are available in the `IdealGasThermo` class. The enthalpy is calculated as

$$H(T) = E_{\mathrm{pot}} + \Delta E_{\mathrm{ZPE}} + \int_0^T C_P \, \mathrm{d}T, \tag{11}$$

  where the constant-pressure heat capacity is

$$C_P = k_{\mathrm{B}} + C_{V,\mathrm{trans}} + C_{V,\mathrm{rot}} + C_{V,\mathrm{vib}}. \tag{12}$$

  The translational heat capacity term is $\frac{3}{2}k_{\mathrm{B}}$, while the rotational term is $\frac{1}{2}k_{\mathrm{B}}$ per rotational DOF. The integral of the vibrational heat capacity takes the same form as in (9), with the sum taken over the vibrational DOFs. The entropy is found from

$$S(T,P) = S^{\circ}_{\mathrm{trans}} + S_{\mathrm{rot}} + S_{\mathrm{vib}} + S_{\mathrm{elec}} - k_{\mathrm{B}} \ln \frac{P}{P^{\circ}}, \tag{13}$$

  where the translational term is calculated at a reference pressure P°. The vibrational component is the same as (10) whereas the remaining components are given below.

$$S_{\mathrm{trans}} = k_{\mathrm{B}} \left\{ \ln \left[\left(\frac{2\pi M k_{\mathrm{B}} T}{h^2} \right)^{3/2} \frac{k_{\mathrm{B}}T}{P^{\circ}} \right] + \frac{5}{2} \right\}, \tag{14}$$

$$S_{\mathrm{rot}} = \begin{cases} 0 & \text{if monatomic,} \\ k_{\mathrm{B}} \left[\ln \left(\dfrac{8\pi^2 I k_{\mathrm{B}} T}{\sigma h^2} \right) + 1 \right] & \text{if linear,} \\ k_{\mathrm{B}} \left\{ \ln \left[\dfrac{\sqrt{\pi I_{\mathrm{A}} I_{\mathrm{B}} I_{\mathrm{C}}}}{\sigma} \left(\dfrac{8\pi^2 k_{\mathrm{B}} T}{h^2} \right)^{3/2} \right] + \dfrac{3}{2} \right\} & \text{if nonlinear,} \end{cases} \tag{15}$$

$$S_{\mathrm{elec}} = k_{\mathrm{B}} \ln \left[2 \times (\text{spin multiplicity}) + 1 \right]. \tag{16}$$

  In the above, I_{A}, I_{B}, and I_{C} are the principal moments of inertia of the molecule, which are calculated in ASE from the atomic coordinates and masses and are available to the user with the `atoms.get_moments_of_inertia` method. In the case of a linear molecule there are two degenerate moments, I. σ is the molecule's symmetry number and h is the Planck constant. Finally, the Gibbs free energy is reported as $G = H - TS$.
- *Crystalline solids.* The vibrational characteristics of a periodic system are often found by treating it as a system of harmonic oscillators (centered around each nucleus), which can be analyzed via the `ase.phonons` module. In this periodic limit, a continuum of vibrational states (described as a phonon density of states) is produced, which can be transformed to a partition function by straightforward means [111]. The `CrystalThermo`

class has methods to calculate the internal energy U and entropy S as below, with the Helmholtz energy also available as $U - TS$:

$$U(T) = E_{\mathrm{pot}} + \Delta E_{\mathrm{ZPE}} + \int_0^\infty \frac{\epsilon}{\mathrm{e}^{\epsilon/k_{\mathrm{B}}T} - 1}\sigma(\epsilon)\mathrm{d}\epsilon, \quad (17)$$

$$S(T) = k_{\mathrm{B}}\int_0^\infty \left[\frac{\epsilon}{k_{\mathrm{B}}T(\mathrm{e}^{\epsilon/k_{\mathrm{B}}T} - 1)} - \ln(1 - \mathrm{e}^{-\epsilon/k_{\mathrm{B}}T})\right]\sigma(\epsilon)\mathrm{d}\epsilon. \quad (18)$$

In this case, the zero-point energy ΔE_{ZPE} is evaluated in integral form, $\int_0^\infty \frac{\epsilon}{2}\sigma(\epsilon)\mathrm{d}\epsilon$.

The integration of methods from the `thermochemistry` module with other ASE methods is straightforward. Typically, one begins by performing a search for a stationary point on the potential energy surface—such as a local minimum or a saddle point—then performs a normal-mode or phonon analysis about this point. The output can be directly fed to the appropriate class in the `thermochemistry` module. However, the user must be careful to assess the validity of the approximate model employed. For example, to calculate the Gibbs free energy of adsorption of a molecule from the gas phase onto a catalytic surface, one might use the approximation:

$$\begin{aligned}\Delta G_{\mathrm{adsorption}} &= G_{\mathrm{mol}} + G_{\mathrm{surf}} - G_{\mathrm{mol+surf}} \\ &\approx G_{\mathrm{mol}}^{\mathrm{ideal\ gas}} - F_{\mathrm{mol}}^{\mathrm{adsorbed,\ harmonic}}\end{aligned} \quad (19)$$

Here G indicates Gibbs free energy while F indicates Helmholtz free energy. Subscripts 'mol' and 'surf' refer to the molecule in the gas phase and the bare catalytic surface, respectively. 'mol + surf' refers to the system composed of the molecule adsorbed on the surface. Two approximations have been employed: first, that the free energy of the surface is unchanged by adsorption of the molecule; second, that the adsorbed molecule has no translational DOFs so the pV term in $G = U - TS + pV$ may be taken to be zero [108] such that $G_{\mathrm{mol}}^{\mathrm{adsorbed,\ harmonic}} = F_{\mathrm{mol}}^{\mathrm{adsorbed,\ harmonic}}$, which may be calculated using the `HarmonicThermo` class.

Below, we show a simple example of computing the ideal gas free energy for an existing `Atoms` object:

```
opt = QuasiNewton(atoms)
opt.run()

vib = Vibrations(atoms)
vib.run()

energy = atoms.get_potential_energy()

thermo = IdealGasThermo(
    vib.get_energies(),
    geometry='nonlinear',
    potentialenergy=energy,
    symmetrynumber=2,
    spin=0.,
    atoms=atoms)
G = thermo.get_gibbs_energy(
temperature=298.0,
pressure=101325.0)
```

8.5. Phase and Pourbaix diagrams

The ASE module `ase.phasediagram` allows to investigate the stability of compounds by means of phase and Pourbaix diagrams.

8.5.1. Phase diagram. A phase stability diagram is obtained by comparing the energy of a particular material with the energies of all possible competing solid structures. For example, the competing phases of a hypothetical compound $KTaO_3$ are K, Ta, and all the possible stoichiometries $K_xTa_yO_z$ (K_2O, KO_3, Ta_2O_5, TaO_3, K_3TaO_8, and so on). The calculated energies of the materials can be used to construct the multidimensional convex hull, which for a given stoichiometry shows which combination of the calculated materials has the lowest energy and is therefore the most stable one at 0K. Depending on the dimensionality (i.e. the number of different elements) the convex hull is composed of lines, planes, or hyperplanes which connect the stable phases. A new material can thus be tested against the convex hull: if its energy is below the hull, the material is stable and it becomes a new point of the hull.

As an example, we investigate the phase stability of $KTaO_3$, a large insulator perovskite material which has potential for running water electrolysis reactions under UV light. After we calculate the total energies of the candidate material and of the possible competing phases with respect to the standard states (in the `references` list), the stability diagram is obtained with the following script:

```
from ase.phasediagram import PhaseDiagram

ref = [('K', 0), ('Ta', 0), ('O2', 0),
       ('K3TaO8', -16.167), ('KO2', -2.288),
       ('KO3', -2.239), ('Ta2O5', -19.801),
       ('TaO3', -8.556), ('TaO', -1.967),
       ('K2O', -3.076), ('K2O2', -4.257),
       ('KTaO3', -13.439)]

pd = PhaseDiagram(ref)
pd.plot()
```

The 3D-phase diagram is shown in figure 10. The stable phases are vertices of the Gibbs triangles and are indicated by green circles, while the unstable phases, which are not vertices, are indicated by red squares. $KTaO_3$ appears stable against phase separation against solid species. Some unstable phases, like KO_3 and K_3TaO_8, have been also included in the list of the possible competing phases. In the present version of the code, it is possible to plot the phase diagram of up to 4 different chemical elements.

8.5.2. Pourbaix diagram. The phase diagram gives information regarding the stability against competing solid phases. The stability against dissolved phases and ions and chemical reactions with water can be addressed by Pourbaix diagrams. The general chemical reaction that we are now considering is

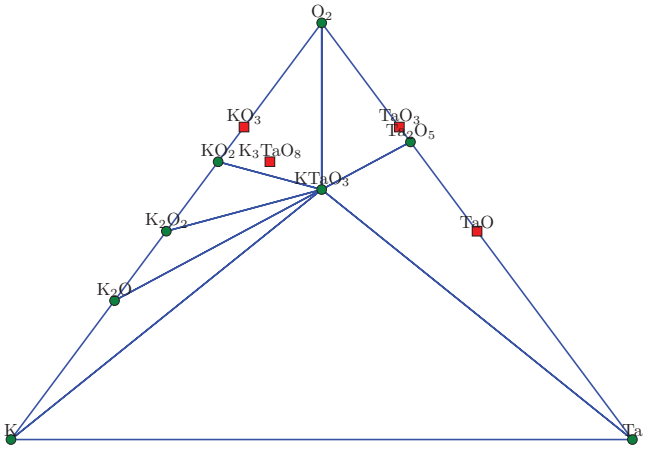


Figure 10. Phase diagram for K-Ta-O. The stable phase are indicated by green circles and are vertices of triangles. The unstable compounds are indicated by red squares.

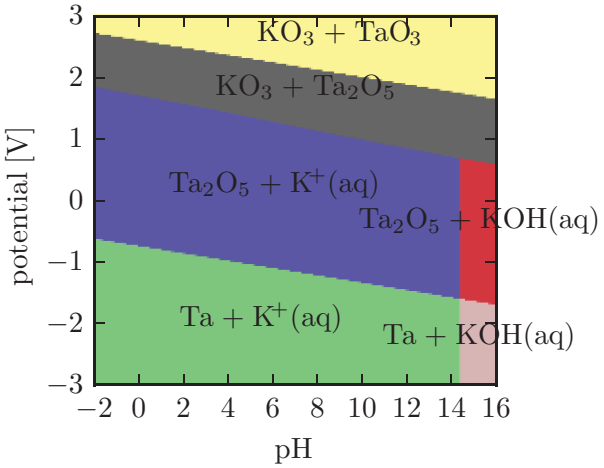


Figure 11. Pourbaix diagram for K-Ta-O.

of the form: $r\mathrm{R} + w\mathrm{H_2O} = p\mathrm{P} + h\mathrm{H}^+ + n\mathrm{e}^-$, where R and P indicate reagents and products with their coefficients r and p, respectively, and the ions $h\mathrm{H}^+$ and $n\mathrm{e}^-$ indicate h dissolved protons and n electrons. The equilibrium potential of the general reaction is calculated using the Nernst equation at 300K:

$$nU = \Delta G + 0.0591 \log \frac{[a_\mathrm{P}]^p}{[a_\mathrm{R}]^r} - 0.0591\ h\ \mathrm{pH}, \qquad (20)$$

where ΔG is the standard free energy of the reaction (in eV), and $[a_\mathrm{P}]$ and $[a_\mathrm{R}]$ are the concentrations of the products and reagents, where the concentrations of the ions are usually equal to 10^{-6} M.

The Nernst equation gives thus a stability diagram as a function of pH and potential. Each diagram is composed of regions divided by straight lines which represent different equilibrium reactions [112]. Depending on the species involved, we can identify three types of lines: (i) vertical lines, where solid and dissolved phases with H^+ ions are involved but without free electrons and the equilibrium is independent of the potential, (ii) horizontal lines, where apart from solid and dissolved substances, free electrons without H^+ ions are present and the equilibrium is this time independent from the pH, (iii) straight lines with 0.0591 h/n slope, where all kind of phases, H^+ ions, and free electrons are present in the Nernst equation. The total energy of the involved species still must be calculated. In this scheme, we calculate the total energies for the solid phases and we include experimental dissolution energies for the ions and the species in aqueous solution from [113, 114]. This method has been described in detail in [115, 116] and recently applied as the stability descriptor to identify novel materials to be used as photoelectrocatalyst in a water splitting cell [117].

To obtain the Pourbaix diagram, ions and dissolved species must be added to the list of reference energies. Around 500 dissolution energies are available in ASE to facilitate this. The Pourbaix diagram (figure 11) is then generated from the list of references in a way similar to the phase diagram above. From the diagram, it appears that $KTaO_3$ is not stable against dissolution in the considered range of pH and potential.

So far, the chemical potentials of the involved chemicals are kept fixed at values corresponding to their standard states. Future releases of the module could include the possibility of tuning the chemical potentials of the elements.

9. Electronic structure

ASE has several tools that support electronic structure calculations and facilitate post processing of calculations. DFT calculators can define common methods which return various properties of the Kohn–Sham states, like eigenvalues, occupation numbers, and the Fermi level. These methods allow for generic calculations of density of states, band gaps, and related quantities using methods in `ase.dft` following any DFT calculation. For calculators that can retrieve the pseudo wavefunction it is also possible to simulate a simple Tersoff–Hamann [118] STM image with the `ase.dft.stm` module.

Calculations with periodic systems usually require Brillouin zone sampling and need a set of k-points on which the Bloch Hamiltonian is diagonalized. The module `ase.dft.kpoints` contains functions that return lists of k-points in a format that is compatible with the electronic structure calculators supported by ASE. For example, one can ask for a Monkhorst–Pack grid [119] or a set of k-points along a path in reciprocal space specified by a set of high symmetry points in the Brillouin zone. The module also contains a dictionary of 'standard paths' [120] in the Brillouin zone for the most common crystal structures. These facilitate systematic calculations of band structures. Below is an example of a script that calculates and plots the Kohn–Sham band structure of bulk Si using ASE's `BandStructure` object. The result is shown in figure 12.

```
from ase.build import bulk
from gpaw import GPAW

si = bulk('Si', 'diamond', a=5.4)
si.calc = GPAW(mode='pw', kpts=(4, 4, 4))
si.get_potential_energy()

si.calc.set(fixdensity=True,
```

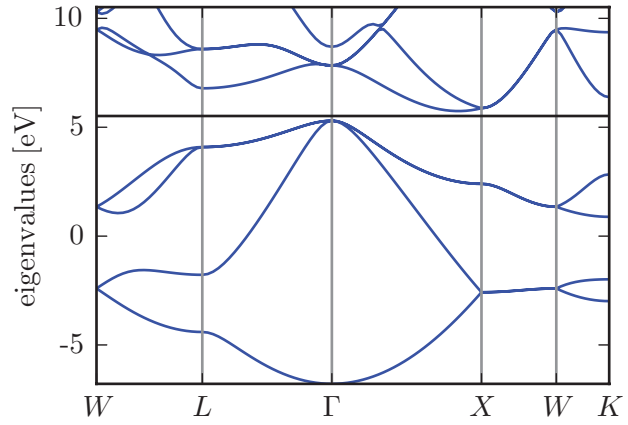


Figure 12. Kohn–Sham band structure of bulk Si.

```
            kpts={'path': 'WLGXWK',
                  'npoints': 100},
            symmetry='off')
si.get_potential_energy()

bs = si.calc.band_structure()
bs.plot(emax=5.0, filename='bands.pdf')
```

9.1. Estimation of exchange–correlation errors

The major approximation within DFT is the exchange–correlation functional. The `BEEFEnsemble` class in ASE provides tools for estimating errors due to the choice of exchange-correlation functional. The most efficient method is tightly linked with the BEEF functionals [121–123]. The BEEF-vdW functional [122], which we will use to explain and exemplify the method, has the functional form

$$E_{\mathrm{xc}} = \sum_{m=0}^{29} a_m E_m^{\mathrm{GGA-x}} + \alpha_c E^{\mathrm{LDA-c}} + (1-\alpha_c) E^{\mathrm{PBE-c}} + E^{\mathrm{nl-c}}, \tag{21}$$

where $E_m^{\mathrm{GGA-x}}$ is GGA exchange with the Legendre polynomial of order m used as enhancement factor. $E^{\mathrm{LDA-c}}$, $E^{\mathrm{PBE-c}}$, and $E^{\mathrm{nl-c}}$ are LDA, PBE, and non-local van der Waals correlation, respectively. The exchange sum expansion coefficients a_m and the correlation weighting parameter α_c are fitted to databases such that the functional performs well for a range of different properties [122, 124]. In addition to the best set of coefficients, which constitutes the main BEEF-vdW functional, an ensemble of functionals with perturbed coefficients are constructed in a procedure inspired by Bayesian statistics, such that the computed ensemble standard deviation for a given calculation is a good estimate of the uncertainty and thus also the potential errors for a range of different properties like reaction enthalpies [122]. The energy of the individual terms in (21) can be parsed from any interfacing calculator to ASE, which contains the ensemble coefficient matrices and generates an energy ensemble using simple multiplication operations without any calls back to the calculator. It is currently implemented for the GPAW and VASP calculators.

Table 2. Enthalpies of reaction.

	Calc. (eV)	Est. error (eV)	Exp.[a] (eV)	Error (eV)
Reaction 1	−1.86	±0.25	−2.14	−0.28
Reaction 2	−1.53	±0.25	−1.80	−0.27
Reaction 3	0.33	±0.02	0.34	0.02

[a] Reference [126].

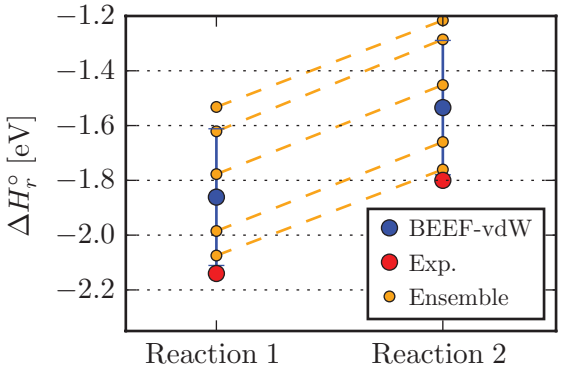


Figure 13. Experimental and calculated enthalpies. The error is well estimated using the ensemble standard deviation. The enthalpy difference between the two reactions is nearly functional independent as illustrated by 5 selected functionals from the ensemble.

As an example, consider two possible reactions in the industrial Fischer–Tropsch process.

$$\mathrm{CO} + 3\ \mathrm{H_2} \rightarrow \mathrm{CH_4} + \mathrm{H_2O} \qquad \text{(Reaction 1)}$$

$$\mathrm{CO} + \frac{5}{2}\ \mathrm{H_2} \rightarrow \frac{1}{2}\ \mathrm{C_2H_6} + \mathrm{H_2O} \qquad \text{(Reaction 2)}$$

The enthalpies of the reactions are calculated and the errors estimated using the ensemble. In addition, the net reaction difference between the two is simultaneously considered as an independent reaction.

$$\mathrm{CH_4} \rightarrow \frac{1}{2}\ \mathrm{C_2H_6} + \frac{1}{2}\ \mathrm{H_2} \qquad \text{(Reaction 3)}$$

Upon comparison with experimental data, see table 2 and figure 13, it is found that the errors versus experimental data and the calculated error estimates are of similar size. Errors and uncertainties within DFT are often systematic, making relative errors much smaller. Considering Reaction 3, both the error and the error estimate are one order of magnitude smaller than for the other reactions, since the errors obtained with any given functional are similar in Reaction 1 and Reaction 2 as illustrated on figure 13.

Another method for testing possible favorable error cancellation to reduce uncertainty without relying on reference data is by establishing correlations in functional dependence for a set of reactions as demonstrated by Christensen *et al* [125]. In addition to error cancellation such analysis can also be used to reveal causes of systematic errors.

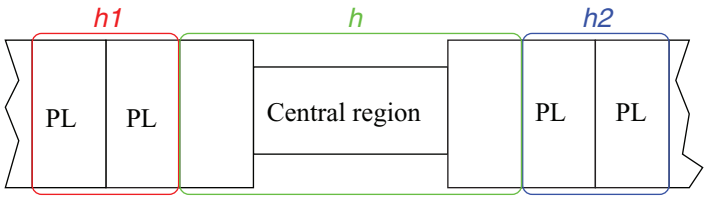


Figure 14. Schematic illustration of the transport setup and the matrices h_1, h and h_2 defining the left lead, central region and right lead, respectively.

9.2. Electron transport

The `ase.transport` module allows for calculating phase-coherent transport within the Landauer–Büttiker picture using Green's functions [127–132]. In the Green's function formalism, the system is divided in to three regions: left lead (L), central region (C), and right lead (R); see figure 14. The semi-infinite leads of the left and right regions are described by self-energies $\Sigma_{L/R}$ and can either be calculated using an efficient decimation technique [133] or approximated in the wide-band approximation. The transmission is calculated from the Green's function expression

$$T(E) = \mathrm{Tr}[G^r(E)\Gamma_L(E)G^a\Gamma^a(E)], \tag{22}$$

where the Green's function is

$$G(z) = (zS - H - \Sigma_L(z) - \Sigma_R(z))^{-1}, \tag{23}$$

where H and S are Hamiltonian and overlap matrices of the central region. The retarded/advanced Green's functions $G^{r/a}$ are obtained by using $z = E \pm i0^+$. From the transmission, the zero-bias conductance is obtained from the expression $G = \frac{2e^2}{h}T(E_F)$, where e, h and E_F are electronic charge, Planck's constant and the Fermi energy, respectively. The effect of a semi-infinite lead α on the central region is described by the self-energy

$$\Sigma_\alpha(E) = (zS_{C\alpha} - H_{C\alpha})g^0_\alpha(z)(zS^\dagger_{C\alpha} - H^\dagger_{C\alpha}), \tag{24}$$

where $g^0_\alpha(z)$ is a surface Green's function that is calculated using a decimation technique [133]. The spectral broadening matrices in (22) are given by $\Gamma_\alpha = i(\Sigma^r_\alpha - \Sigma^a_\alpha)$.

In the transport module, the C, L and R regions (figure 14) are specified by three Hamiltonian matrices h, h_1 and h_2 (in case of non-orthogonal basis the three overlap matrices s, s_1 and s_2 are also needed). h_1 and h_2 should contain two principal layers; a principal layer (PL) being a periodic part of the Hamiltonian such that there is only coupling between nearest-neighbour principal layers. By default, the transport module assumes that the coupling between the central region and lead α, $h_{C\alpha}$, is the same as the coupling between principal layers. In this case, h should contain at least one principal layer of lead 1 in its upper left corner, and at least one principal layer of lead 2 in the lower right corner. See figure 14.

With the transport module, a number of standard transport properties can be calulated, such as:

- transmission function
- eigenchannel-resolved transmission functions
- eigenchannel wavefunctions
- projected density of states

and there are various tools for analysis of the transport mechanism:

- rotation to eigenbasis of a subspace, e.g. basis functions on the molecule
- cutting out a subspace, e.g. to see which molecular states are involved in the transport

To illustrate how to use the transport module we consider a simple example consisting of a junction between graphene leads with zig-zag edges and a benzene molecule linked via alkyne groups; see figure 15(a). For simplicity we describe the π-system only using a Hückel model Hamiltonian with all nearest-neighbour hopping elements set to -3 eV and on-site energies set to zero, i.e. bond length alternation is not taken into account. In this model, the principal layers can be chosen as illustrated in figure 15(a) and contain 20 carbon atoms. Periodic boundary conditions are used in the plane perpendicular to the transport direction and the 1D Brillouin zone is sampled using 201 k-points. A transport calculation may look like:

```
calc = TransportCalculator(h=h, h1=h1,
    h2=h2, energies=energies)
transm = calc.get_transmission()
plt.plot(energies, transm)
```

In the first line, the transport calculator is set up from the Hamiltonians and a list of energies for the transmission calculation. Finally, the transmission curve is plotted using matplotlib.

Figure 15(b) shows the transmission for the junction in the pristine graphene system obtained using the Hückel model. The pristine graphene system uses the same supercell size perpendicular to the transport direction. Additionally, the transmission obtained by taking the Hamiltonian and overlap matrices from a GPAW DFT calculation using a single-zeta local atomic orbital basis [22, 134] set is shown (Junction*). We note that the Hückel and DFT results are in good agreement. The main difference is that the particle–hole transmission symmetry in the Hückel model caused by the system being an alternant hydrocarbon is broken slightly in the DFT calculation.

The transmission of the pristine graphene system goes to zero at the Fermi level since the density of states goes to zero here; graphene can be viewed as zero-bandgap semiconductor. When considering the transmission of the molecular junction, it is expected to always be below the pristine system. Interestingly, the transmission at around ± 0.2 eV has a peak which approaches

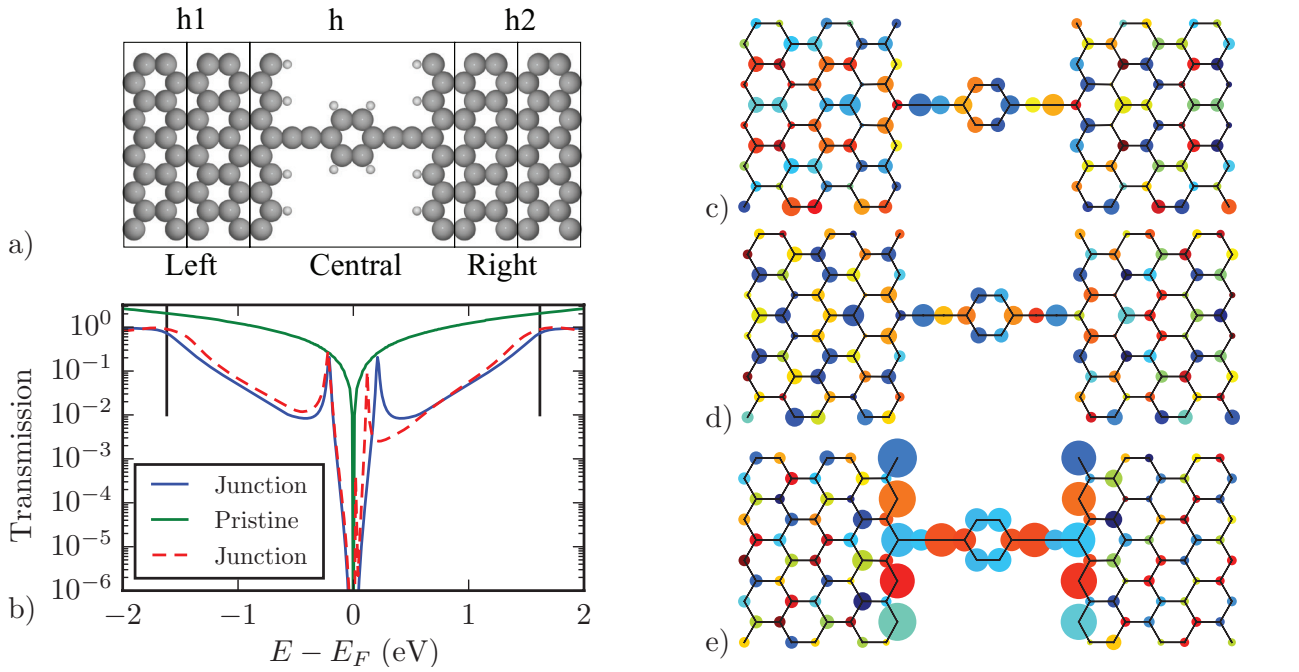


Figure 15. (a) Structure of the junction. (b) Transmission of pristine graphene and of the molecular junction calculated using Hückel theory. The transmission from a DFT calculation is also shown for comparison. The vertical lines indicate the molecular states obtained by subdiagonalizing the molecular subspace. (c)–(e) Left eigenchannel scattering state at an energy corresponding to the molecular HOMO level, LUMO level and −0.2 eV, respectively. For (c) and (d) it is for the Γ k-point, while it is for $k = 0.26$ Å^{-1} in (e).

the transmission of the pristine system. This indicates that the charge carriers are not scattered at all at these energies.

To understand the origin of the broad transmission peaks around ±1.6 eV, indicated by the black vertical lines, we can use the analysis functions provided by the transport calculator. we have diagonalized the subspace of the molecule including linkers. This yields the eigenenergies of the subsystem, and from this we find that the transmission peak position coincides with the HOMO and LUMO energy.

To further investigate the origin of the transmission peaks we can calculate the eigenchannel scattering functions. The eigenchannels are calculated using the method of [135]. Figures 15(c) and (d) visualize the eigenchannel states using coloured circles on the atomic sites; the size of the circle indicates the absolute weight and the color indicates the phase.

When considering the eigenchannel calculated at the HOMO and LUMO energy found above, see figures 15(c) and (d), it resembles the isolated molecule HOMO and LUMO (not shown) to a large extent. This indicates resonance transmission through the HOMO and LUMO. The eigenchannel state in figure 15(e) has considerable weight on the edges of the graphene leads, which indicates a transport mechanism involving edge states. It is well known that graphene zigzag edges have localized edge states at the Dirac point [136].

10. Additional features

10.1. Command-line tools

In addition to the Python library that can be used in Python scripts and programs, an ASE installation will also include four command-line tools for work on the command line:

- `ase gui`: Graphical user interface
- `ase db`: Manipulate ASE-databases
- `ase build`: Build molecules or crystals
- `ase run`: Run simple calculation

The GUI can be used to visualize, manipulate and analyze trajectories. Among the analysis tools are the possiblity to create x–y plots from the structure and energetics.

The `ase db` command-line tool is for querying and manipulating databases as described in detail in section 4.1.

Here is an example showing how to use the `ase build` and `ase run` tools to optimize the structure of the H_2 molecule with the PBE functional:

```
$ ase build H2|ase run nwchem -p xc=PBE -f .02
Running: H2
       Step     Time          Energy         fmax
LBFGS:    0  15:04:19     -31.487747     0.6220
LBFGS:    1  15:04:19     -31.492828     0.0378
LBFGS:    2  15:04:19     -31.492848     0.0020
```

The `ase build` and `ase run` tools are very useful for doing quick test calculations, but to get the most use out of ASE you will need to write scripts that can take advantage of the flexibility of the Python language.

10.2. Projects using ASE

A number of the calculators listed in table 1 are developed externally to ASE. For examples, Atomistica [137] is a library of classical and tight-binding interatomic potentials that can be compiled as a Python extension module and directly interfaced with ASE. It provides implementations of empirical bond-order potentials (Abell–Tersoff–Brenner [138–140], REBO2 [141], etc), their screened counterparts [142, 143], and

embedded atom method potentials [144]. The non-orthogonal tight-binding module supports charge-selfconsistency [145] and can read a variety of parameter databases. The `quippy` package is also fully interoperable with ASE. It provides a calculator for the interatomic potentials implemented in the QUIP Fortran code [44] such as the Gaussian Approximation Potential [146] as well as support for multiscale QM/MM calculations [147].

The `matscipy` package [148], developed by Lars Pastewka and James Kermode, builds on top of ASE and adds a number of additional general purpose tools for computational materials science, including a fast neighbour list implemented in C, tools for structure analysis, post processing and visualisation, a calculator capable of communicating with remote codes over POSIX sockets, as well as a number of specialist modules targeting specific applications including fracture and contact mechanics.

10.3. Technical information

ASE is free software licensed under GNU LGPLv2.1 or newer. The source code and documentation are available from the ASE web page https://wiki.fysik.dtu.dk/ase.

ASE requires Python 2 or 3 and the NumPy package. SciPy [19] and MatPlotLib [20] is recommended. The graphical user interface requires PyGTK [149].

Install ASE by running `pip install ase` (or `pip3 install ase` for Python 3 users) or see the web page for alternative methods. The `ase-users` mailing list or `#ase` IRC channel on Freenode.net [150] are available for support.

Everybody is invited to participate in using and developing the code. Coordination and discussions take place on the mailing list whereas the development takes place on GitLab, at https://gitlab.com/ase/ase. Please send us bug-reports, patches, code, and ideas.

11. Conclusions

The atomic simulation environment is currently in use by many researchers working on diverse applications within condensed matter physics, chemistry, and other fields. There are also quite a few developers adding new features to the environment. So what can be expected from ASE in the future?

At the present stage the core structure of ASE dealing with the basic properties of atoms and calculators is well developed, and it is therefore straightforward to add new functionality. For example, new interfaces to the ONETEP [151] and Amber [152] codes are currently being implemented. In the near future we hope to see further development in several areas. To mention some of them: (1) New screening or search methods (like the currently implemented genetic algorithm) using machine learning could be implemented. (2) The analysis of the electronic information obtained by DFT calculations could be significantly improved. Recently a way of calculating band structures was introduced but there is a general need for more detailed analysis based on electronic spectra, densities, or wave functions. (3) Crystal symmetry analysis is currently done mostly by the DFT codes. This task could be done directly by ASE. (4) Currently some of the external codes like CP2K and GPAW are 'kept alive' when atoms are moved to avoid computational overhead when restarting a calculation. This could be implemented in a more generic way in ASE so that other calculators easily could obtain this feature. (5) The fairly new database module allows for storage and retrieval of DFT calculations. The database can be used to keep track of the status of many similar calculations performed for example in a computational screening study. However, there are currently no utility functions or classes to perform this task in an easy way.

The users of ASE benefit from the large number of functions available for setting up, controlling, and analyzing simulations based on many different calculators, and from the large flexibility in the Python language itself to construct loops and to allow for interplay between different simulations. As the number of available calculators increases and new functionality is added, ASE will hopefully become an even more attractive toolbox contributing to efficient development and utilization of electronic structure theory and molecular dynamics simulations. We hope that ASE will also encourage and contribute to further collaborative efforts with open exchange of not only data and results but also efficient scripting to the benefit of the research community.

Acknowledgments

The authors acknowledge funding from: The European Union's Horizon 2020 research and innovation program under grant agreement no. 676580 with The Novel Materials Discovery (NOMAD) Laboratory, a European Center of Excellence; research grant 9455 from VILLUM FONDEN; Deutsche Forschungsgemeinschaft (grant PA2023/2); the UK Engineering and Physical Sciences Research Council (grants EP/L014742/1, EP/L027682/1 and EP/P002188/1).

References

[1] Jones R 2015 Density functional theory: its origins, rise to prominence and future *Rev. Mod. Phys.* **87** 897–923
[2] Quantum chemistry and solid-state physics software https://en.wikipedia.org/wiki/List_of_quantum_chemistry_and_solid-state_physics_software
[3] Knowledgebase of interatomic models https://openkim.org
[4] Dubois P F 1994 Making applications programmable *Comput. Phys.* **8** 70–3
[5] Gamma E, Helm R, Johnson R and Vlissides J 1995 *Design Patterns: Elements of Reusable Object-Oriented Software* (Reading, MA: Addison-Wesley)
[6] Bahn S R and Jacobsen K W 2002 An object-oriented scripting interface to a legacy electronic structure code *Comput. Sci. Eng.* **4** 56–66
[7] DACAPO code https://wiki.fysik.dtu.dk/dacapo
[8] NumPy www.numpy.org
[9] Mills G and Jónsson H 1994 Quantum and thermal effects in H_2 dissociative adsorption: evaluation of free energy barriers in multidimensional quantum systems *Phys. Rev. Lett.* **72** 1124–7

[10] Jónsson H, Mills G and Jacobsen K W 1998 Nudged elastic band method for finding minimum energy paths of transitions *Classical and Quantum Dynamics in Condensed Phased Simulations, Proc. Int. School of Physics 'Computer Simulation of Rare Events and Dynamics of Classical and Quantum Condensed-Phased Systems' (Lerici, Villa Marigola, 7 July–18 July 1997) ed* B J Berne *et al* (Singapore: World Scientific) pp 385–404

[11] Khorshidi A and Peterson A A 2016 Amp: a modular approach to machine learning in atomistic simulations *Comput. Phys. Commun.* **207** 310–24

[12] Oliphant T E 2007 Python for scientific computing *Comput. Sci. Eng.* **9** 10–20

[13] Millman K J and Aivazis M 2011 Python for scientists and engineers *Comput. Sci. Eng.* **13** 9–12

[14] Walt S V D, Colbert S C and Varoquaux G 2011 The numpy array: a structure for efficient numerical computation *Comput. Sci. Eng.* **13** 22–30

[15] Kendall R A *et al* 2000 High performance computational chemistry: an overview of NWChem a distributed parallel application *Comput. Phys. Commun.* **128** 260–83

[16] Valiev M *et al* 2010 NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations *Comput. Phys. Commun.* **181** 1477–89

[17] Perdew J P, Burke K and Ernzerhof M 1996 Generalized gradient approximation made simple *Phys. Rev. Lett.* **77** 3865–8

[18] Nocedal J and Wright S J 2006 *Numerical Optimization* (*Springer Series in Operations Research and Financial Engineering*) (Berlin: Springer)

[19] SciPy http://scipy.org

[20] matplotlib http://matplotlib.org/

[21] Mortensen J J, Hansen L B and Jacobsen K W 2005 Real-space grid implementation of the projector augmented wave method *Phys. Rev.* B **71** 035109

[22] Enkovaara J *et al* 2010 Electronic structure calculations with GPAW: a real-space implementation of the projector augmented-wave method *J. Phys.: Condens. Matter* **22** 253202

[23] Gonze X *et al* 2009 ABINIT: First-principles approach to material and nanosystem properties *Comput. Phys. Commun.* **180** 2582–615

[24] Clark S J, Segall M D, Pickard C J, Hasnip P J, Probert M I, Refson K and Payne M C 2005 First principles methods using CASTEP *Z. Kristallogr.—Cryst. Mater.* **220** 567–70

[25] Hutter J, Iannuzzi M, Schiffmann F and VandeVondele J 2014 CP2K: atomistic simulations of condensed matter systems *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **4** 15–25

[26] Köster A M and deMon Developers 2006 deMon2k

[27] Aradi B, Hourahine B and Frauenheim T 2007 DFTB+, a sparse matrix-based implementation of the DFTB method *J. Phys. Chem.* A **111** 5678–84

[28] Daw M S and Baskes M I 1983 Semiempirical, quantum mechanical calculation of hydrogen embrittlement in metals *Phys. Rev. Lett.* **50** 1285–8

[29] Jacobsen K W, Stoltze P and Nørskov J K 1996 A semi-empirical effective medium theory for metals and alloys *Surf. Sci.* **366** 394–402

[30] Gulans A, Kontur S, Meisenbichler C, Nabok D, Pavone P, Rigamonti S, Sagmeister S, Werner U and Draxl C 2014 Exciting: a full-potential all-electron package implementing density-functional theory and many-body perturbation theory *J. Phys.: Condens. Matter* **26** 363202

[31] Blum V, Gehrke R, Hanke F, Havu P, Havu V, Ren X, Reuter K and Scheffler M 2009 *Ab initio* molecular simulations with numeric atom-centered orbitals *Comput. Phys. Commun.* **180** 2175–96

[32] Wimmer E, Krakauer H, Weinert M and Freeman A J 1981 Full-potential self-consistent linearized-augmented-plane-wave method for calculating the electronic structure of molecules and surfaces: O_2 molecule *Phys. Rev.* B **24** 864–75

[33] Frisch M J *et al* 2004 *Gaussian 03, Revision C.02* (Wallingford, CT: Gaussian, Inc.)

[34] Pronk S *et al* 2013 GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit *Bioinformatics* **29** 845–54

[35] Koskinen V M P 2009 Density-functional tight-binding for beginners *Comput. Mater. Sci.* **47** 237

[36] Ismail-Beigi S and Arias T 2000 New algebraic formulation of density functional calculation *Comput. Phys. Commun.* **128** 1–45

[37] Plimpton S 1995 Fast parallel algorithms for short-range molecular dynamics *J. Comput. Phys.* **117** 1–19

[38] Jones J E 1924 On the determination of molecular fields. II. From crystal measurements and kinetic theory data *Proc. R. Soc. Lond. A* **106** 709–18

[39] Stewart J J P 1990 MOPAC: A semiempirical molecular orbital program *J. Comput.-Aided Mol. Des.* **4** 1–103

[40] Morse P M 1929 Diatomic molecules according to the wave mechanics. II. Vibrational levels *Phys. Rev.* **34** 57–64

[41] Andrade X *et al* 2015 Real-space grids and the Octopus code as tools for the development of new simulation approaches for electronic systems *Phys. Chem. Chem. Phys.* **17** 31371–96

[42] Tadmor E B, Elliott R S, Sethna J P, Miller R E and Becker C A 2011 The potential of atomistic simulations and the knowledgebase of interatomic models *JOM* **63** 17

[43] Giannozzi P *et al* 2009 QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials *J. Phys.: Condens. Matter* **21** 395502

[44] Csányi G, Winfield S, Kermode J R, de Vita A, Comisso A, Bernstein N and Payne M C 2007 *Expressive Programming for Computational Physics in Fortran 95* (Bristol: Institute of Physics Publishing)

[45] Soler J M, Artacho E, Gale J D, Garc A, Junquera J, Ordej P and Sánchez-Portal D 2002 The SIESTA method for *ab initio* order-*N* materials simulation *J. Phys.: Condens. Matter* **14** 2745

[46] Jorgensen W L, Chandrasekhar J, Madura J D, Impey R W and Klein M L 1983 Comparison of simple potential functions for simulating liquid water *J. Chem. Phys.* **79** 926–35

[47] Furche F, Ahlrichs R, Hättig C, Klopper W, Sierka M and Weigend F 2014 Turbomole *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **4** 91–100

[48] Kresse G and Furthmüller J 1996 Efficient iterative schemes for *ab initio* total-energy calculations using a plane-wave basis set *Phys. Rev.* B **54** 11169–86

[49] Tkatchenko A and Scheffler M 2009 Accurate molecular van der Waals interactions from ground-state electron density and free-atom reference data *Phys. Rev. Lett.* **102** 073005

[50] Grimme S, Antony J, Ehrlich S and Krieg H 2010 A consistent and accurate *ab initio* parametrization of density functional dispersion correction (DFT-D) for the 94 elements H–Pu *J. Chem. Phys.* **132** 154104

[51] Grimme S, Ehrlich S and Goerigk L 2011 Effect of the damping function in dispersion corrected density functional theory *J. Comput. Chem.* **32** 1456–65

[52] Grimme D3 ASE calculator https://gitlab.com/ehermes/ased3

[53] Haunschild R and Klopper W 2012 New accurate reference energies for the G2/97 test set *J. Chem. Phys.* **136** 164102

[54] Hahn T 2005 *International Tables for Crystallography, Space-Group Symmetry* (*International Tables for Crystallography*) (New York: Wiley)

[55] Tripa C E, Zubkov T S, Yates J T, Mavrikakis M and Nørskov J K 1999 Molecular N_2 chemisorption-specific

adsorption on step defect sites on Pt surfaces *J. Chem. Phys.* **111** 8651–8
[56] Freund H-J and Pacchioni G 2008 Oxide ultra-thin films on metals: new materials for the design of supported metal catalysts *Chem. Soc. Rev.* **37** 2224–42
[57] Xie S *et al* 2014 Atomic layer-by-layer deposition of Pt on Pd nanocubes for catalysts with enhanced activity and durability toward oxygen reduction atomic layer-by-layer deposition of Pt on Pd nanocubes for catalysts with enhanced activity and durability toward oxygen *Nano Lett.* **14** 3570–6
[58] Munroe R, xkcd https://xkcd.com/927/
[59] POV-Ray www.povray.org/
[60] Alchagirov A B, Perdew J P, Boettger J C, Albers R C and Fiolhais C 2003 Reply to 'Comment on 'Energy and pressure versus volume: equations of state motivated by the stabilized jellium model'' *Phys. Rev.* B **67** 026103
[61] Yu W, Wu J and Panda D 2004 Fast and scalable startup of MPI programs in InfiniBand clusters *High Performance Computing—HIPC 2004: 11th Int. Conf. on High Performance Computing (Bangalore, India, 19–22 December 2004) (Lecture Notes in Computer Science* vol 3296) ed L Bouge and V K Prasanna (Berlin: Springer) pp 440–9
[62] CP2K code https://www.cp2k.org
[63] 2013 Standard for information technology portable operating system interface (POSIX(R)) base specifications, issue 7, IEEE Std 1003.1, 2013 Edition (incorporates IEEE Std 1003.1-2008 and IEEE Std 1003.1-2008/Cor 1-2013) pp 1–3906
[64] Frenkel D and Smit B 2002 *Understanding Molecular Simulation: from Algorithms to Applications* (London: Academic)
[65] Berendsen H J C, Postma J P M, van Gunsteren W F, DiNola A and Haak J R 1984 Molecular dynamics with coupling to an external bath *J. Chem. Phys.* **81** 3684
[66] Allen M and Tildesley D 1989 *Computer Simulation of Liquids* (Oxford: Clarendon)
[67] Hoover W G 1985 Canonical dynamics—equilibrium phase-space distributions *Phys. Rev.* A **31** 1695–7
[68] Hoover W G 1986 Constant-pressure equations of motion *Phys. Rev.* A **34** 2499–500
[69] Parrinello M and Rahman A 1980 Crystal-structure and pair potentials—a molecular-dynamics study *Phys. Rev. Lett.* **45** 1196–9
[70] Parrinello M and Rahman A 1981 Polymorphic transitions in single-crystals—a new molecular-dynamics method *J. Appl. Phys.* **52** 7182–90
[71] Melchionna S 2000 Erratum: Constrained systems and statistical distribution (2000 *Phys. Rev. E* 61 6165) *Phys. Rev.* E **62** 5864–4
[72] Melchionna S 2000 Constrained systems and statistical distribution *Phys. Rev.* E **61** 6165–70
[73] Press W, Teukolsky S, Vetterling W and Flannery B 1992 *Numerical Recipes in* C (Cambridge: Cambridge University Press)
[74] Nocedal J 1980 Updating quasi-Newton matrices with limited storage *Math. Comput.* **35** 773–82
[75] Liu D C and Nocedal J 1989 On the limited memory bfgs method for large scale optimization *Math. Prog.* **45** 503–28
[76] Bitzek E, Koskinen P, Gähler F, Moseler M and Gumbsch P 2006 Structural relaxation made simple *Phys. Rev. Lett.* **97** 170201
[77] Packwood D, Kermode J, Mones L, Bernstein N, Woolley J, Gould N, Ortner C and Csányi G 2016 A universal preconditioner for simulating condensed phase materials *J. Chem. Phys.* **144** 164109
[78] Peterson A A 2014 Global optimization of adsorbate—surface structures while preserving molecular identity *Top. Catal.* **57** 40–53
[79] Henkelman G and Jónsson H 2000 Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points *J. Chem. Phys.* **113** 9978–85
[80] Henkelman G, Uberuaga B P and Jónsson H 2000 A climbing image nudged elastic band method for finding saddle points and minimum energy paths *J. Chem. Phys.* **113** 9901–4
[81] Smidstrup S, Pedersen A, Stokbro K and Jónsson H 2014 Improved initial guess for minimum energy path calculations *J. Chem. Phys.* **140** 214106
[82] Kolsbjerg E L, Groves M M and Hammer B 2016 An automated nudged elastic band method *J. Chem. Phys.* **145** 094107
[83] Rasmussen A M H, Groves M N and Hammer B 2014 Remote activation of chemical bonds in heterogeneous catalysis *ACS Catal.* **4** 1182–8
[84] Goubert G, Rasmussen A M H, Dong Y, Groves M N, McBreen P H and Hammer B 2014 Walking-like diffusion of two-footed asymmetric aromatic adsorbates on Pt(1 1 1) *Surf. Sci.* **629** 123–31
[85] Kolsbjerg E L, Groves M N and Hammer B 2016 Pyridine adsorption and diffusion on Pt(1 1 1) investigated with density functional theory *J. Chem. Phys.* **144** 164112
[86] Melander M, Laasonen K and Jónsson H 2015 Removing external degrees of freedom from transition-state search methods using quaternions *J. Chem. Theory Comput.* **11** 1055–62
[87] Pedersen A, Hafstein S F and Jónsson H 2011 Efficient sampling of saddle points with the minimum-mode following method *SIAM J. Sci. Comput.* **33** 633–52
[88] Wales D J and Doye J P K 1997 Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms *J. Phys. Chem.* A **101** 5111–6
[89] Goedecker S 2004 Minima hopping: an efficient search method for the global minimum of the potential energy surface of complex molecular systems *J. Chem. Phys.* **120** 9911–7
[90] Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)
[91] Forrest S 1993 Genetic algorithms: principles of natural selection applied to computation *Science* **261** 872–8
[92] Johnston R 2003 Evolving better nanoparticles: genetic algorithms for optimising cluster geometries *Dalton Trans.* **22** 4193–207
[93] Glass C W, Oganov A R and Hansen N 2006 USPEX—Evolutionary crystal structure prediction *Comput. Phys. Commun.* **175** 713–20
[94] Jóhannesson G, Bligaard T, Ruban A, Skriver H, Jacobsen K and Nørskov J 2002 Combined electronic structure and evolutionary search approach to materials design *Phys. Rev. Lett.* **88** 255506
[95] Lysgaard S, Landis D D, Bligaard T and Vegge T 2014 Genetic algorithm procreation operators for alloy nanoparticle catalysts *Top. Catal.* **57** 33–9
[96] Lysgaard S, Mýrdal J S G, Hansen H A and Vegge T 2015 A DFT-based genetic algorithm search for AuCu nanoalloy electrocatalysts for CO_2 reduction *Phys. Chem. Chem. Phys.* **17** 28270–6
[97] Vilhelmsen L and Hammer B 2012 Systematic study of Au_6 to Au_{12} gold clusters on MgO(1 0 0) F centers using density-functional theory *Phys. Rev. Lett.* **108** 126101

[98] Deaven D M and Ho K M 1995 Molecular-geometry optimization with a genetic algorithm *Phys. Rev. Lett.* **75** 288–91

[99] Vilhelmsen L B and Hammer B 2014 A genetic algorithm for first principles global structure optimization of supported nano structures *J. Chem. Phys.* **141** 044711

[100] Jensen P B, Lysgaard S, Quaade U J and Vegge T 2014 Designing mixed metal halide ammines for ammonia storage using density functional theory and genetic algorithms *Phys. Chem. Chem. Phys.* **16** 19732–40

[101] Jensen P B, Bialy A, Blanchard D, Lysgaard S, Reumert A K, Quaade U J and Vegge T 2015 Accelerated DFT-based design of materials for ammonia storage *Chem. Mater.* **27** 4552–61

[102] Frederiksen T, Paulsson M, Brandbyge M and Jauho A -P 2007 Inelastic transport theory from first-principles: methodology and applications for nanoscale devices *Phys. Rev.* B **75** 205413

[103] Porezag D and Pederson M R 1996 Infrared intensities and raman-scattering activities within density-functional theory *Phys. Rev.* B **54** 7830–6

[104] Profeta M and Mauri F 2001 Theory of resonant raman scattering of tetrahedral amorphous carbon *Phys. Rev.* B **63** 245415

[105] Dove M 1993 *Introduction to Lattice Dynamics, Cambridge Topics in Mineral Physics and Chemistry* (Cambridge: Cambridge University Press)

[106] Alf D 2009 PHON: A program to calculate phonons using the small displacement method *Comput. Phys. Commun.* **180** 2622–33

[107] Baroni S, Giannozzi P and Testa A 1987 Green's-function approach to linear response in solids *Phys. Rev. Lett.* **58** 1861–4

[108] Jones G *et al* 2008 First principles calculations and experimental insight into methane steam reforming over transition metal catalysts *J. Catal.* **259** 147–60

[109] Hill T L 1960 *An Introduction to Statistical Thermodynamics* (Reading, MA: Addison-Wesley)

[110] Cramer C J 2004 *Essentials of Computational Chemistry* 2nd edn (New York: Wiley)

[111] McQuarrie D A 2000 *Statistical Mechanics* (Sausalito: University Science Books)

[112] Verink E D 2011 *Simplified Procedure for Constructing Pourbaix Diagrams* (New York: Wiley)

[113] Johnson J W, Oelkers E H and Helgeson H C 1992 SUPCRT92: A software package for calculating the standard molal thermodynamic properties of minerals, gases, aqueous species and reactions from 1 to 5000 bar and 0 to 1000 °C *Comput. Geosci.* **18** 899–947

[114] Pourbaix M 1966 *Atlas of Electrochemical Equilibria in Aqueous Solutions* 1st edn (Oxford: Pergamon)

[115] Persson K A, Waldwick B, Lazic P and Ceder G 2012 Prediction of solid–aqueous equilibria: Scheme to combine first-principles calculations of solids with experimental aqueous states *Phys. Rev.* B **85** 235438

[116] Castelli I E, Thygesen K S and Jacobsen K W 2013 Calculated pourbaix diagrams of cubic perovskites for water splitting: stability against corrosion *Top. Catal.* **57** 265–72

[117] Castelli I E, Hüser F, Pandey M, Li H, Thygesen K S, Seger B, Jain A, Persson K A, Ceder G and Jacobsen K W 2015 New light-harvesting materials using accurate and efficient bandgap calculations *Adv. Energy Mater.* **5** 1400915

[118] Tersoff J and Hamann D R 1985 Theory of the scanning tunneling microscope *Phys. Rev.* B **31** 805–13

[119] Monkhorst H J and Pack J D 1976 Special points for Brillouin-zone integrations *Phys. Rev.* B **13** 5188–92

[120] Setyawan W and Curtarolo S 2010 High-throughput electronic band structure calculations: challenges and tools *Comput. Mater. Sci.* **49** 299–312

[121] Mortensen J J, Kaasbjerg K, Frederiksen S L, Nørskov J K, Sethna J P and Jacobsen K W 2005 Bayesian error estimation in density-functional theory *Phys. Rev. Lett.* **95** 216401

[122] Wellendorff J, Lundgaard K T, Møgelhøj A, Petzold V, Landis D D, Nørskov J K, Bligaard T and Jacobsen K W 2012 Density functionals for surface science: Exchange–correlation model development with Bayesian error estimation *Phys. Rev.* B **85** 235149

[123] Wellendorff J, Lundgaard K T, Jacobsen K W and Bligaard T 2014 mBEEF: An accurate semi-local Bayesian error estimation density functional *J. Chem. Phys.* **140** 144107

[124] Wellendorff J, Silbaugh T L, Garcia-Pintos D, Nørskov J K, Bligaard T, Studt F and Campbell C T 2015 A benchmark database for adsorption bond energies to transition metal surfaces and comparison to selected DFT functionals *Surf. Sci.* **640** 36–44 (Reactivity concepts at surfaces: coupling theory with experiment)

[125] Christensen R, Hansen H A and Vegge T 2015 Identifying systematic DFT errors in catalytic reactions *Catal. Sci. Technol.* **5** 4946–9

[126] Linstrom P J and Mallard W G (ed) 2005 *NIST Chemistry WebBook* (*NIST Standard Reference Database* vol 69) (Washington, DC: National Institute of Standards and Technology)

[127] Landauer R 1957 Spatial variation of currents and fields due to localized scatterers in metallic conduction *IBM J. Res. Dev.* **1** 223–31

[128] Landauer R 1970 Electrical resistance of disordered one-dimensional lattices *Phil. Mag.* **21** 863–7

[129] Caroli C, Combescot R, Nozieres P and Saint-James D 1971 Direct calculation of the tunneling current *J. Phys. C: Solid State Phys.* **4** 916

[130] Meir Y and Wingreen N S 1992 Landauer formula for the current through an interacting electron region *Phys. Rev. Lett.* **68** 2512–5

[131] Brandbyge M, Mozos J-L, Ordejón P, Taylor J and Stokbro K 2002 Density-functional method for nonequilibrium electron transport *Phys. Rev.* B **65** 165401

[132] Strange M, Kristensen I S, Thygesen K S and Jacobsen K W 2008 Benchmark density functional theory calculations for nanoscale conductance *J. Chem. Phys.* **128** 114714

[133] Guinea F, Tejedor C, Flores F and Louis E 1983 Effective two-dimensional Hamiltonian at surfaces *Phys. Rev.* B **28** 4397–402

[134] Larsen A H, Vanin M, Mortensen J J, Thygesen K S and Jacobsen K W 2009 Localized atomic basis set in the projector augmented wave method *Phys. Rev.* B **80** 195112

[135] Paulsson M and Brandbyge M 2007 Transmission eigenchannels from nonequilibrium Green's functions *Phys. Rev.* B **76** 115117

[136] Nakada K, Fujita M, Dresselhaus G and Dresselhaus M S 1996 Edge state in graphene ribbons: nanometer size effect and edge shape dependence *Phys. Rev.* B **54** 17954–61

[137] Atomistica https://github.com/Atomistica/atomistica

[138] Abell G C 1985 Empirical chemical pseudopotential theory of molecular and metallic bonding *Phys. Rev.* B **31** 6184–96

[139] Tersoff J 1986 New empirical model for the structural properties of silicon *Phys. Rev. Lett.* **56** 632–5

[140] Brenner D W 1990 Empirical potential for hydrocarbons for use in simulating the chemical vapor deposition of diamond films *Phys. Rev.* B **42** 9458–71

[141] Brenner D W, Shenderova O A, Harrison J A, Stuart S J, Ni B and Sinnott S B 2002 A second-generation reactive

empirical bond order (REBO) potential energy expression for hydrocarbons *J. Phys.: Condens. Matter* **14** 783–802
[142] Pastewka L, Mrovec M, Moseler M and Gumbsch P 2012 Bond order potentials for fracture, wear and plasticity *MRS Bull.* **37** 493–503
[143] Pastewka L, Klemenz A, Gumbsch P and Moseler M 2013 Screened empirical bond-order potentials for Si–C *Phys. Rev.* B **87** 205410
[144] Foiles S M and Baskes M I 2012 Contributions of the embedded-atom method to materials science and engineering *MRS Bull.* **37** 485–91
[145] Elstner M, Porezag D, Jungnickel G, Elsner J, Haugk M, Frauenheim T, Suhai S and Seifert G 1998 Self-consistent-charge density-functional tight-binding method for simulations of complex materials properties *Phys. Rev.* B **58** 7260–8
[146] Bartók A P, Payne M C, Kondor R and Csányi G 2010 Gaussian approximation potentials: the accuracy of quantum mechanics, without the electrons *Phys. Rev. Lett.* **104** 136403
[147] Bernstein N, Kermode J R and Csányi G 2009 Hybrid atomistic simulation methods for materials systems *Rep. Prog. Phys.* **72** 026501
[148] Matscipy https://gitlab.com/libAtoms/matscipy
[149] Hunter J D 2007 Matplotlib: A 2D graphics environment *Comput. Sci. Eng.* **9** 90–5
[150] freenode https://freenode.net/
[151] Skylaris C -K, Haynes P D, Mostofi A A and Payne M C 2005 Introducing ONETEP: Linear-scaling density functional simulations on parallel computers *J. Chem. Phys.* **122** 084119
[152] Salomon-Ferrer R, Case D A and Walker R C 2012 An overview of the Amber biomolecular package *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **3** 198–210