libvdwxc — a scalable library for van der Waals density functionals

Ask Hjorth Larsen[1], Mikael Kuisma[2], Yann Pouillon[3], Joakim Löfgren[4], Paul Erhart[4], Per Hyldgaard[5]

[1] Nano-bio Spectroscopy Group and ETSF Scientific development centre, Departamento de Física de Materiales, Universidad del País Vasco, Spain
[2] Department of Chemistry, University of Jyväskylä, Finland
[3] Universidad de Cantabria, Spain
[4] Department of Applied Physics, Chalmers University of Technology, Sweden
[5] Department of Microtechnology and Nanoscience, Chalmers University of Technology, Sweden

January 8, 2019

What is libvdwxc?

- Small library for evaluating the non-local energy and its derivatives for functionals in the vdW-DF family (vdW-DF1, vdW-DF2)
- libvdwxc allows electronic structure codes to evaluate these functionals in combination with libxc
- Written in C
- `https://libvdwxc.org`
- Supports the recent spin-polarized vdW formalism
  T. Thonhauser *et al*, Phys.Rev.Lett. **115**, 136402 (2015)
- Currently two codes can use libvdwxc: GPAW and Octopus
- Dependencies: FFTW; optionally MPI, FFTW-MPI, PFFT
- License: GPL (for now)

van der Waals functionals

Form

- LDA correlation + GGA exchange + vdW correlation:

$$E_{\mathrm{xc}}^{\mathrm{vdW}} = E_{\mathrm{c}}^{\mathrm{LDA}}[n] + E_{x}^{\mathrm{GGA}}[n] + E_{\mathrm{c}}^{\mathrm{NL}}[n]$$

- Non-local term is:

$$E_{\mathrm{c}}^{\mathrm{NL}}[n] = \frac{1}{2} \iint n(\mathbf{r}) \phi[n](\mathbf{r}, \mathbf{r}') n(\mathbf{r}) \, \mathrm{d}\mathbf{r} \, \mathrm{d}\mathbf{r}'$$

- (So the vdW-DF functionals are true non-local density functionals, and not “corrections”)
- M. Dion, H. Rydberg, E. Schröder, D. C. Langreth, and B. I. Lundqvist, Phys. Rev. Lett. **92**, 246401 (2004)
- K. Berland and P. Hyldgaard, Phys. Rev. B **89**, 035412 (2014)

(Our own) motivations for libvdwxc

- Improve vdW support in the GPAW code
- Old vdW-DF implementation in GPAW was not scalable beyond 20 cores
- Test on molecule from S22 test set in large box showed 75%+ time spent doing `MPI_Gather`
- Add vdW support to Octopus
- Different codes have different vdW implementations, numerics, van der Waals kernel parametrizations, making comparisons difficult

Why another library?

- The existing library Libxc is for local and semilocal density functionals
- Libxc calculates energy and derivatives pointwise
- Libxc also provides special semilocal functionals used by the vdW-DF family
- But: Non-local functionals require infrastructure which is not in Libxc — in our case vdW kernel, spline interpolation, FFTs, ...
- Also: Must be designed for scalability
- Similar library: GridXC (talk by Alberto García later)

Evaluating vdW functionals

Consider again:

$$E_{\mathrm{c}}^{\mathrm{NL}}[n] = \frac{1}{2} \iint n(\mathbf{r}) \phi[n](\mathbf{r}, \mathbf{r}') n(\mathbf{r}) \, \mathrm{d}\mathbf{r} \, \mathrm{d}\mathbf{r}'$$

- The kernel has the form $\phi(q_0(\mathbf{r}), q_0(\mathbf{r}'), \|\mathbf{r} - \mathbf{r}'\|)$
- q_0 is a GGA-like quantity: $q_0(\mathbf{r}) = q_0(n(\mathbf{r}), \|\boldsymbol{\nabla} n(\mathbf{r})\|)$
- The kernel can be pre-parametrized and stored in a file
- During a calculation, the double space integral must still be evaluated
- The double space integral is expensive but can be approximated as a single integral using the method by Román-Pérez and Soler

The Román-Pérez–Soler method

- Discretize q_0 to 20-point grid (typically) and expand the kernel in $20 \times 20 = 400$ terms

$$\phi(q_0, q_0', d) = \sum_{\alpha,\beta=1}^{20} \phi_{\alpha\beta}(d) p_\alpha(q_0) p_\beta(q_0').$$

- p_α are fixed auxiliary functions to interpolate values on the coarse q_0 grid
- Let $\theta_\alpha(\mathbf{r}) = n(\mathbf{r}) p_\alpha(q_0(\mathbf{r}))$, and the energy becomes a convolution

$$\begin{aligned} E_{\mathrm{c}}^{\mathrm{nl}} &= \frac{1}{2} \sum_{\alpha\beta} \iint \theta_\alpha(\mathbf{r}) \phi_{\alpha\beta}(\|\mathbf{r} - \mathbf{r}'\|) \theta_\beta(\mathbf{r}') \, \mathrm{d}\mathbf{r} \, \mathrm{d}\mathbf{r}', \\ &= \frac{1}{2} \sum_{\alpha\beta} \int \theta_\alpha^*(\mathbf{k}) \theta_\beta(\mathbf{k}) \phi_{\alpha\beta}(k) \, \mathrm{d}\mathbf{k}. \end{aligned}$$

Full algorithm

- Calculate $q_0(\mathbf{r})$ and $\theta_\alpha(\mathbf{r})$
- Transform $\theta_\alpha(\mathbf{r})$ to Fourier space $\rightarrow \theta_\alpha(\mathbf{k})$
- Calculate energy from convolution as well as derivative

$$F_\alpha(\mathbf{k}) = \sum_\beta \theta_\beta(\mathbf{k}) \phi_{\alpha\beta}(k)$$

- Transform $F_\alpha(\mathbf{k})$ back to $F_\alpha(\mathbf{r})$
- Evaluate potential using $F_\alpha(\mathbf{r})$

libvdwxc expects the density on a uniform real-space 3D grid.
Fourier transforms are the most expensive operation.

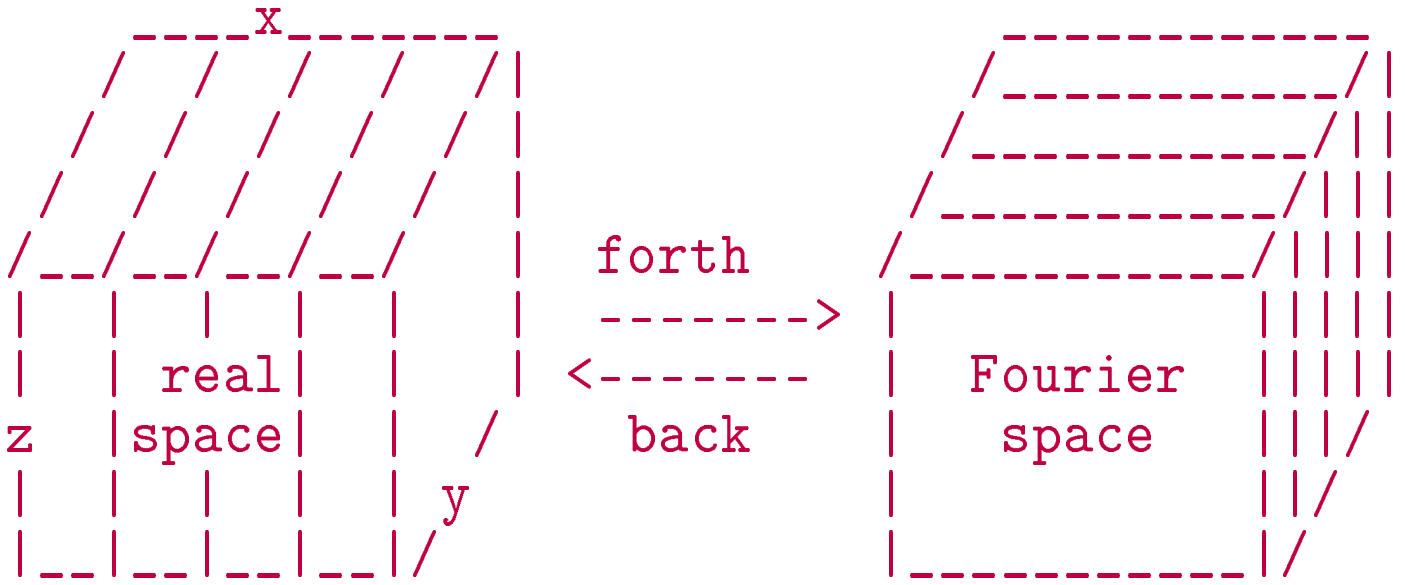
Parallel 3D Fourier transforms with FFTW3-MPI

Strategy

$$\widehat{f}(\mathbf{k}) = \iiint f(\mathbf{x}) \exp(-i\mathbf{k}\cdot\mathbf{x}) \,\mathrm{d}x_1 \,\mathrm{d}x_2 \,\mathrm{d}x_3$$

- A single FFT does not parallelize well
- What we really have is a 3D array of 1D transforms
- Different cores can do different (but whole) 1D transforms
- Luckily we can use FFTW-MPI for all this
- (Remaining operations are local!)

Parallel Fourier transforms



Algorithm

- Get data into block distribution over x only
- Take Fourier transform over y and z
- Transform to block distribution over y (parallel transpose)
- Fourier transform over x

Performance tricks

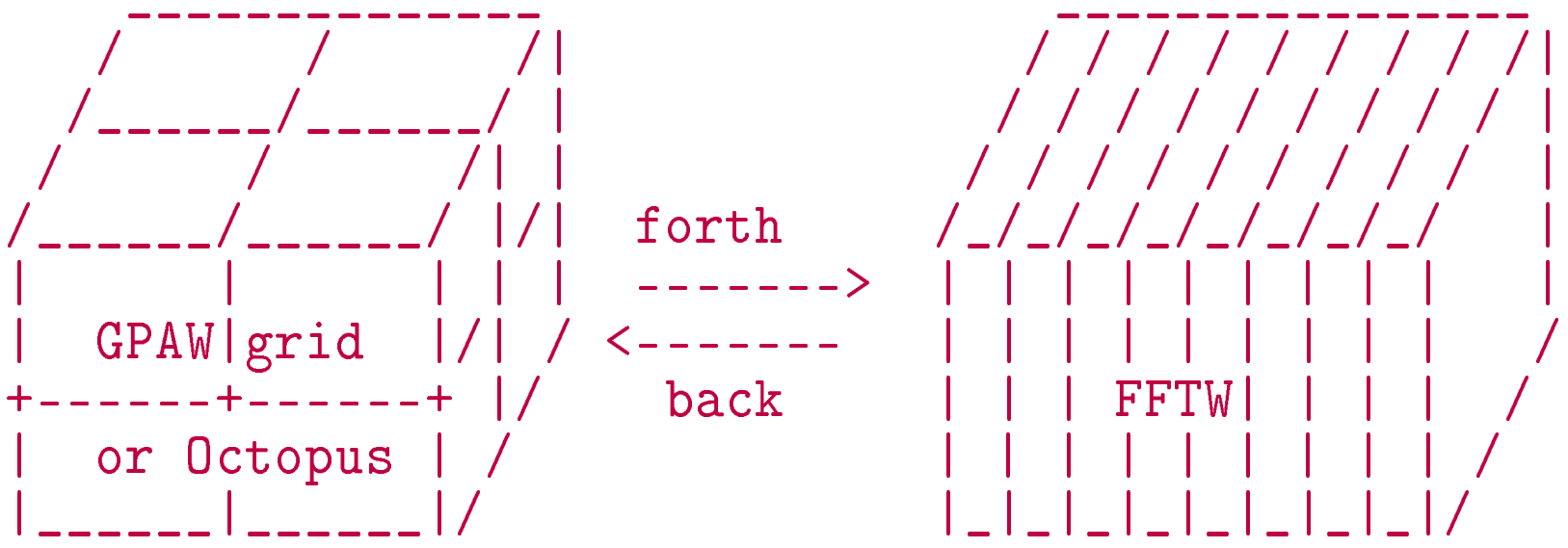
- FFT output goes into input buffer
- $\theta_\alpha(\mathbf{r})$ and friends are *strided* (α axis is contiguous, not $\mathbf{r}$)
- Work on transposed output
- In general, write non-silly loops (mind memory order, buffers)

(None of this is particularly advanced)

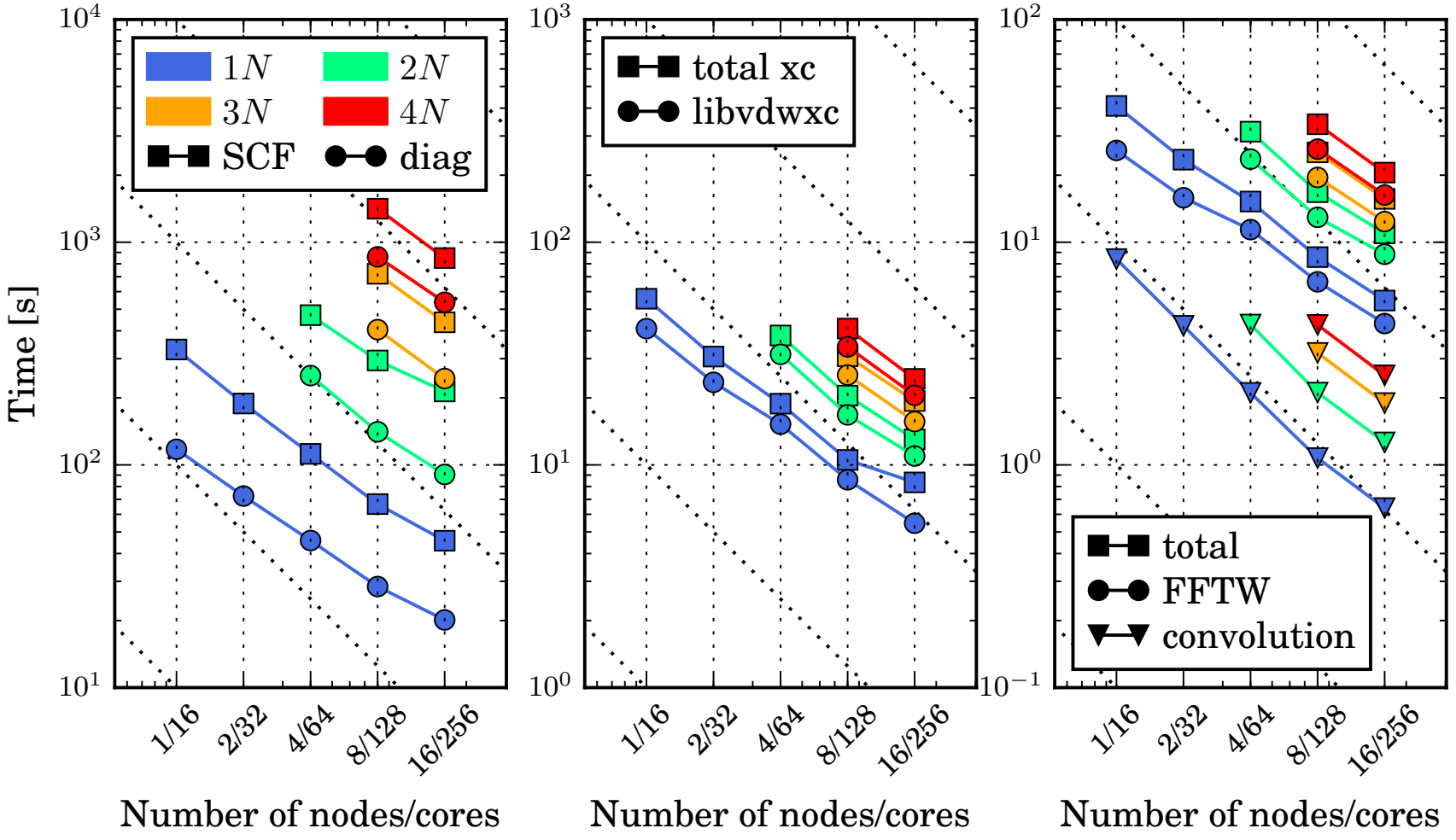
2D versus 3D distributions

- libvdwxc can also use PFFT which distributes over two axes at a time
- More scalable
- But less efficient for realistically sized DFT grids in our tests, due to extra redistribution step

Distribution from GPAW/Octopus to FFT format



Octopus can also distribute from irregular boxes.



- Test with Au144 + organic ligands (2500 atoms x [1,2,3,4]) and GPAW LCAO mode
- 1) Full scf step, 2) XC timings including redistribution, 3) time spent in libvdwxc

Challenges and future work

- Most important: Get codes to interface libvdwxc instead of rolling their own
- Support pluggable kernels
- Facilitate data redistribution for DFT code
- Easier installation: Solve MKL/FFTW linking problem (MKL is evil)
- (Also: Add Vydrov–van Voorhis functional)

libvdwxc

- Web page: https://libvdwxc.org
- Development: https://gitlab.com/libvdwxc/libvdwxc
- Features: Non-local energy of vdW-DF family of functionals: (vdW-DF, vdW-DF2, vdW-DF-CX)
- Requirements: FFTW3 or FFTW3-MPI + MPI
- Uses kernel parametrization from Quantum Espresso (so far!)
- A.H. Larsen, M. Kuisma, J. Löfgren, Y. Pouillon, P. Erhart, and P. Hyldgaard:
  Modelling Simul. Mater. Sci. Eng. **25** 065004, 2017.
  "libvdwxc: a library for exchange–correlation functionals in the vdW-DF family"

The Atomic Simulation Environment

ASE is a free (LGPLv2.1+) toolkit to set up and control atomistic calculations in a fully scripted environment using Python.

Main features

- The `Atoms` object: A collection of atoms
- Calculators: Capable of calculating energies and forces of atoms, often using an external code as backend
- Algorithms working with atoms/calculators: Structure optimization, molecular dynamics, basin hopping, minima hopping, nudged elastic band, ...
- Many utilities: Build crystals, surfaces, ...
- Read/write structures in many formats
- Also: GUI, command-line utilities

Example: Structure optimization with Espresso

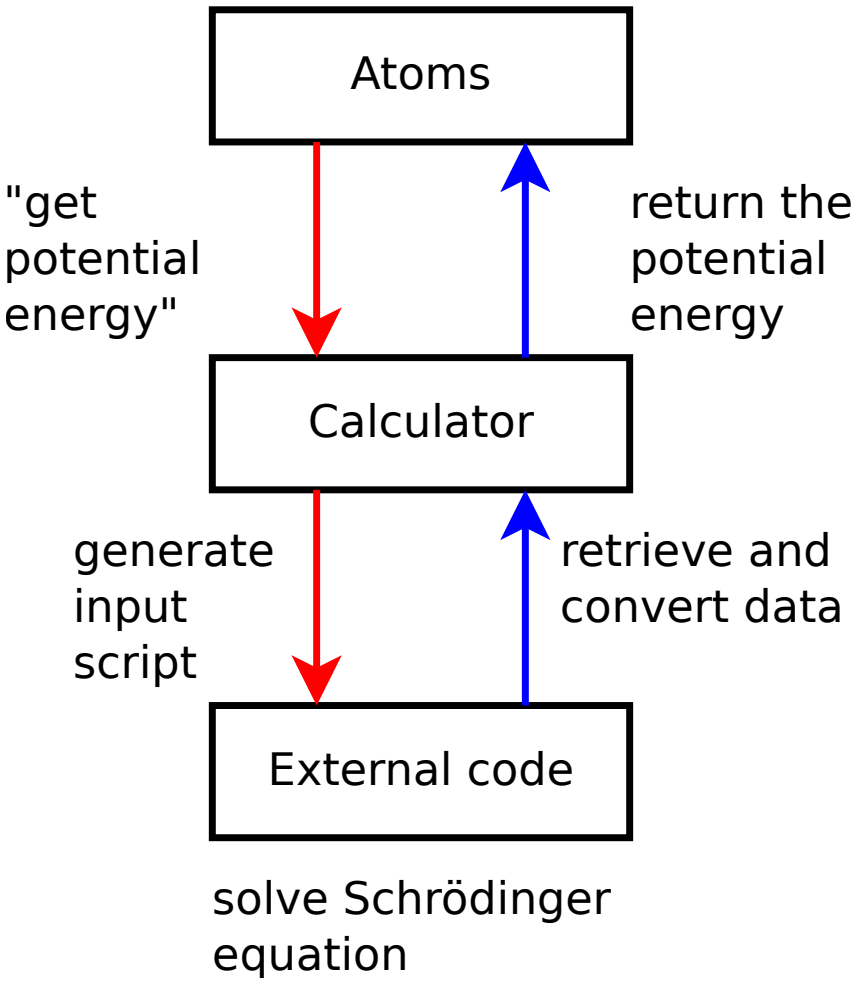
```
from ase import Atoms
from ase.optimize import BFGS
from ase.calculators.espresso import Espresso

atoms = Atoms('H2O', positions=[[-1, 0, 0],
                                [1, 0, 0],
                                [0, 0, 1]])
atoms.center(vacuum=3.0)
atoms.calc = Espresso(
    ecutwfc=40., pseudo_dir='.', tprnfor=True,
    pseudopotentials={'H': 'H_ONCV_PBE-1.0.upf',
                      'O': 'O_ONCV_PBE-1.0.upf'})
opt = BFGS(system, trajectory='opt.traj',
           logfile='opt.log')
opt.run(fmax=0.05)
```

Codes with ASE calculators

ASAP	Abinit	Atomistica
CP2K	Castep	DFTB+
Dacapo	ELK	Exciting
FHI-aims	Fleur	GPAW
Gaussian	Gromacs	Hotbit
JDFTx	LAMMPS	MOPAC
NWChem	Octopus	OpenKIM
OpenMX	QUIP	Quantum Espresso
Siesta	Turbomole	VASP
deMon	matscipy	

Communication with external codes



1) One process per calculation

- ASE creates inputfile, runs code in subprocess (see figure)

2) Persistent subprocess

- External process remains alive over multiple calculations
- IO uses sockets, pipes or files

3) Within same process

- Direct access to functions, data
- Requires Python bindings

Sockets in ASE using the i-PI protocol

- i-PI is a "universal force engine" specifying a protocol for communicating atomic positions, forces, etc., over sockets
- Idea is to run a driver from server, while code runs as a client
- `http://ipi-code.org/`
- ASE implements the i-PI protocol
- ASE launches server which can be used as a calculator, `ase.calculators.socketio.SocketIOCalculator`
- The SocketIOCalculator wraps another (ordinary) calculator and can automatically launch the client process (for Quantum Espresso, Siesta, FHI-aims)
- ASE can also run as a client

```
from ase.build import molecule
from ase.calculators.siesta import Siesta
from ase.optimize import BFGS
from ase.calculators.socketio import SocketIOCalculator

socket = 'mysocket'
fdf_arguments = {'MD.TypeOfRun': 'Master',
                 'Master.code': 'i-pi',
                 'Master.interface': 'socket',
                 'Master.address': socket,
                 'Master.socketType': 'unix'}

atoms = molecule('H2O', vacuum=3.0)
atoms.rattle(stdev=0.1)
siesta = Siesta(fdf_arguments=fdf_arguments)
opt = BFGS(atoms, trajectory='opt.siesta.traj',
           logfile='opt.siesta.log')
with SocketIOCalculator(siesta, unixsocket=socket) as calc:
    atoms.calc = calc
    opt.run(fmax=0.05)
```

ASE

- Web page: `https://wiki.fysik.dtu.dk/ase/`
- Gitlab: `https://gitlab.com/ase/ase`
- Mailing lists, IRC: `https://wiki.fysik.dtu.dk/ase/contact.html`
- A.H. Larsen *et al* 2017 *J. Phys.: Condens. Matter* **29** 273002. "The atomic simulation environment—a Python library for working with atoms"