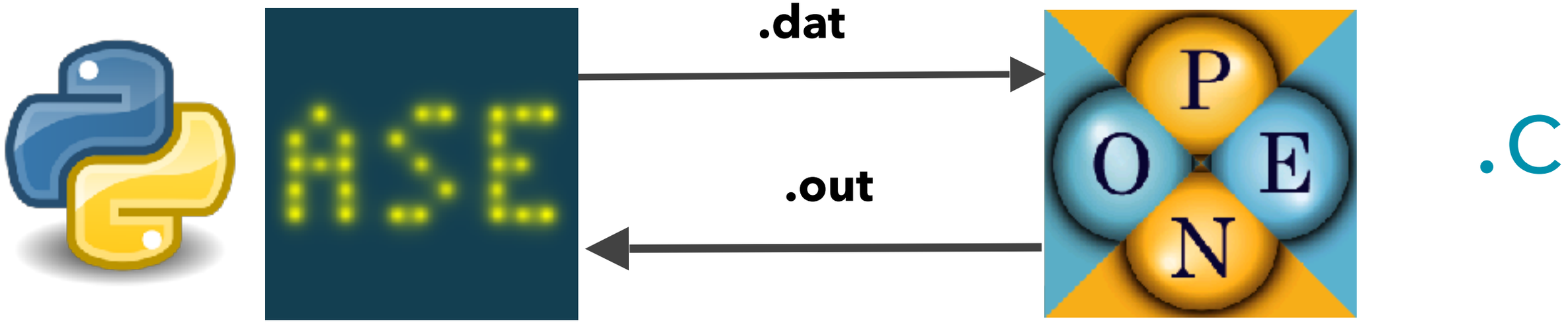


An ASE Calculator for OpenMX:

providing automation of running OpenMX

Charles Johnson* and Jaejun Yu

Seoul National University

*Current Address: Imperial College London

2016.11.24

2nd OpenMX Developer's Meeting, KAIST, 2016.11.23-25

https://wiki.fysik.dtu.dk/ase/

Atomic Simulation Environment

a set of tools and Python modules for
setting up, manipulating, running, visualizing and analyzing atomistic simulations

```
>>> # Example: structure optimization of hydrogen molecule
>>> 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
```

Modules

ase (Atom)	ase (Atoms)	build	calculators
collections	constraints	db	dft
data	ga	geometry	gui
io	lattice	md	neb
neighborlist	optimize	parallel	phasediagram
phonons	spacegroup	transport	thermochemistry
units	utils	vibrations	visualize

Supported calculators:

abinit Asap Atomistica CASTEP CP2K deMon DFTB ELK EMT fleur GPAW! GROMACS Hotbit JACAPO LAMMPS NWCHEM siesta TURBOMOLE VASP amber Gaussian Grimme-D3 Mopac tip3p

Motivation

- ASE (atomic simulation environment) is a convenient and generic python package for representing atoms.
- ASE connects with many "**Calculators**" including VASP but not OpenMX until now.
- Example python script:

```
from ase.atoms import Atoms
from ase.calculators.openmx import OpenMX


atoms = Atoms(**kwargs)  # Specifies the species and geometry
calc = OpenMX(**kwargs)  # Specifies the method of solving for the system
atoms.set_calculator(calc)  # Methods of Atoms objects can now use OpenMX
calc.calculate(atoms=atoms) # writes input file, executes OpenMX and
                            # reads results from .out file
total_energy = atoms.get_total_energy()
```

Features of the calculator

- Automatically writes a .dat input file using the keyword arguments of the calculator for OpenMX, executes and reads results from the .out file.
- Molecular dynamics can be turned on. This will automatically update the atoms object.
- Spin polarised systems: collinear or noncollinear (with constraint) methods are supported.
- (Partial) Density of States calculations as well as Band structure can be set-up and plots can be automatically generated.
- If VESTA is installed, visualising 3D molecular orbitals can be achieved using `OpenMX.get_mo()` method.
- ASE uses eV, Å and s as units. You may use other units from the ase.units module.

To get OpenMX to work with ASE

- Need to have OpenMX installed and openmx and mpirun executables in PATH and...

- Either set the environment variables

```
OPENMX_COMMAND (e.g. 'mpirun -np 4 openmx ./%s -nt 2 > ./%s')
OPENMX_VPS_PATH (directory containing the pseudo-potential files)
OPENMX_PAO_PATH (directory containing the pseudo-atomic orbital files)
```

- Or specify the number of processes and threads and directories of VPS and PAO through the calculator arguments:

```
calc = OpenMX(processes=4, threads=2, vps_path=~/lib/DFT_DATA13/VPS, pao_path=~/lib/DFT_DATA13/PAO)
```

Specifying the PAO, VPS and orbitals to use

- Default settings are specified as default_dictionary in `default_settings.py`.
- These can be overridden by specifying `dft_data_dict` in the keyword arguments.

```
from atoms.units import Bohr


data = {
 ‘Si’: {
     ‘cutoff radius’: 8 * Bohr  # will find nearest available cutoff
     ‘orbitals used’: [2, 2, 1]  # [‘s’, ‘p’, 'd’,...]
     ‘pseudo-potential suffix’: ‘’  # e.g. if VPS file is
# Si_CA13_SCH2p, then pseudo-potential suffix must be ‘_SCH2p’
}
}
calc = OpenMX(dft_data_dict=data)
```

A spin non-polarized, total energy calculation

- If no arguments are specified, then OpenMX will use these defaults for the self-consistent field calculation:

```
From atoms.units import Ry, Ha

calc = OpenMX(label=”openmx”, xc=”PBE”, kgrid=(1, 1, 1),
energy_cutoff=150*Ry, electronic_temperature=300,
scf_max_iter=40, scf_criterion=1e-6*Ha,
eigenvalue_solver=”Cluster”, mixing_type=’RMM-DIIS’,
init_mixing_weight=0.3, max_mixing_weight=0.4,
mixing_history=5, mixing_start_pulay=6)
```

- Refer to OpenMX manual version 3.8 -> input files -> keywords in the SCF section for explanation of these parameters:

For collinear spin polarised systems

- If `initial_spin` (guess of the net spin of each atom in units of ħ) is specified then spin polarisation will be turned on in OpenMX. `hubbard_u_values` should be specified as well (default to 0 for each orbital). `hubbard_occupation` can be ‘dual’, ‘onsite’ or ‘full’, default is ‘dual’.

```
u_values = {
        ‘Mn’: {
          ‘1d’: 4.0  # eV
        }
}
calc = OpenMX(initial_spin=[1.5, -0.5], hubbard_u_values=u_values,
hubbard_occupation=‘dual’)
```

For noncollinear spin polarised systems

- If `spin_euler_angle` (guess of spin alignment for each atom) is specified, OpenMX will solve the system for noncollinear spin. To apply a constraint, `nc_spin_constraint_penalty` (energy prefactor of penalty functional) should be specified with `orbital_euler_angle` (the confinement alignment).

```
calc = OpenMX(spin_euler_angle=[(0., 0.), (45., 0.)],
orbital_euler_angle=[(15., 30.), (10., 0.)],

nc_spin_constraint_penalty=1.  # eV)
```

Molecular Dynamics / Geometry Optimization

- If `md_type` is specified, OpenMX will perform molecular dynamics calculations. `md_type` can be `‘Opt’, 'NVE', 'NVT_VS' or 'NVT_NH'`.

```
from atoms.units import fs, Ha, Bohr

calc = OpenMX(md_type=’Opt’, md_max_iter=1,
time_step=0.5*fs,
md_criterion=1e-4*Ha/Bohr)
```

- After the molecular dynamics calculation, the atoms object’s positions automatically update to the final geometry.

(Partial) Density of States

- To calculate the necessary information of the density of states, specify `dos_erange` as an argument.

```
calc = OpenMX(dos_erange=(-10, 10), dos_kgrid=(5, 5, 5))
```

- After the calculator has solved the system, add the following line to either display or save to a file a plot of the (partial) density of states.

```
calc.get_dos(method='Tetrahedron', pdos=True,
atom_index_list=[1], orbital_list=[''], file_format='pdf')
```

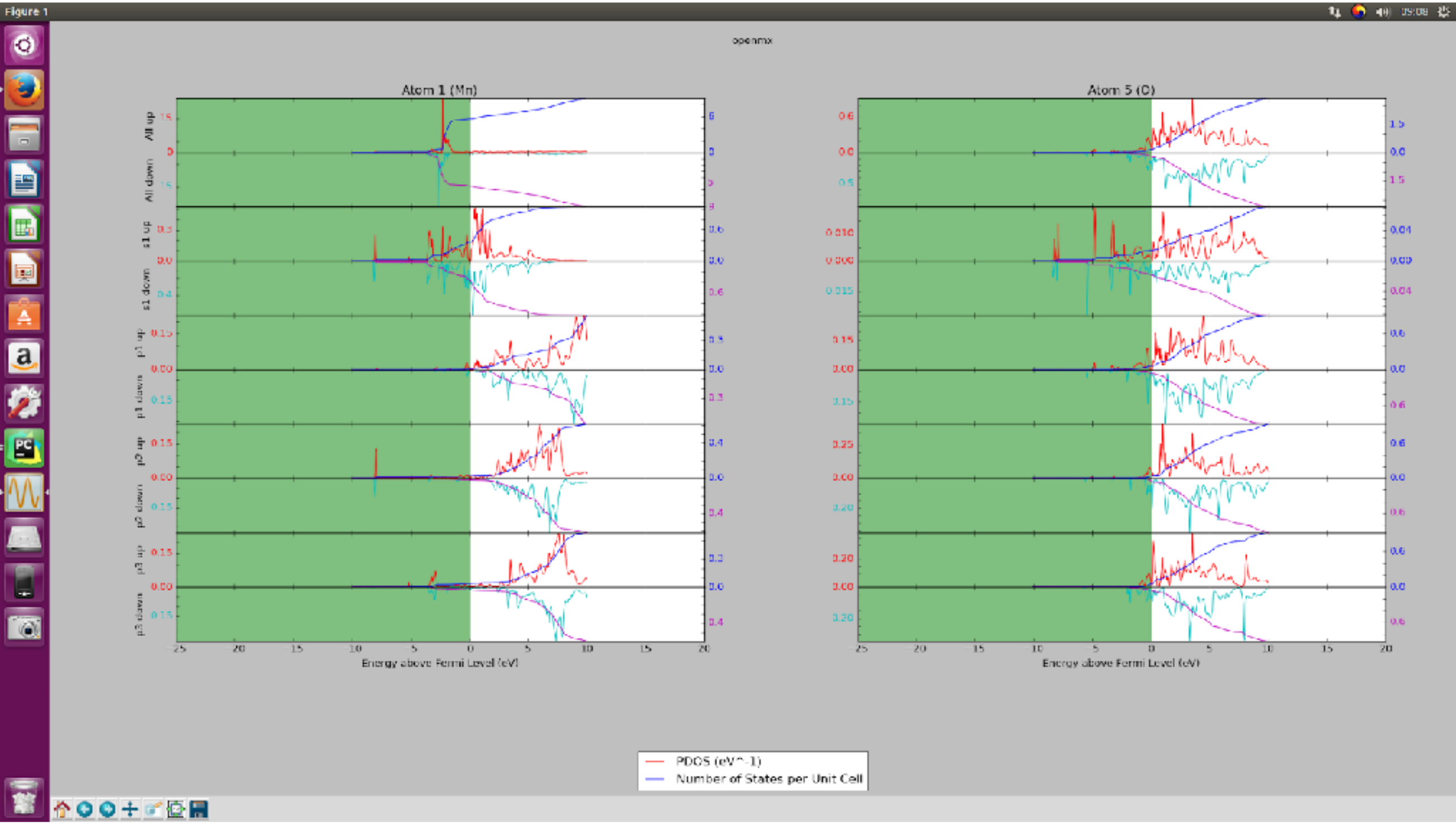


Figure 1
openmx
Atom 1 (Mn)
Atom 5 (C)
All up
All down
s1 up
s1 down
p1 up
p1 down
p2 up
p2 down
p3 up
p3 down
Energy above Fermi Level (eV)
Energy above Fermi Level (eV)
PDOS (eV^-1)
Number of States per Unit Cell

Band Structure

- To calculate band structure, set `band_dispersion=True` and specify kpath.

```
kpath = [{'kpts': 20, 'start_point': (0., 0., 0.), 'end_point': (1., 0., 0.),
          'path_symbols': ('g', 'X')},
         {'kpts': 20, 'start_point': (1., 0., 0.), 'end_point': (1., 0.5,
0.),
          'path_symbols': ('X', 'W')}]
unit_cell = [(1, 0, 0), (0, 1, 0), (0, 0, 1)]
calc = OpenMX(band_dispersion=True, band_kpath=kpath,
              band_kpath_unitcell=unit_cell)
```

- To display or save to a file a plot of the band structure:

```
calc.get_band(erange=(-10, 10), file_format='pdf')
```

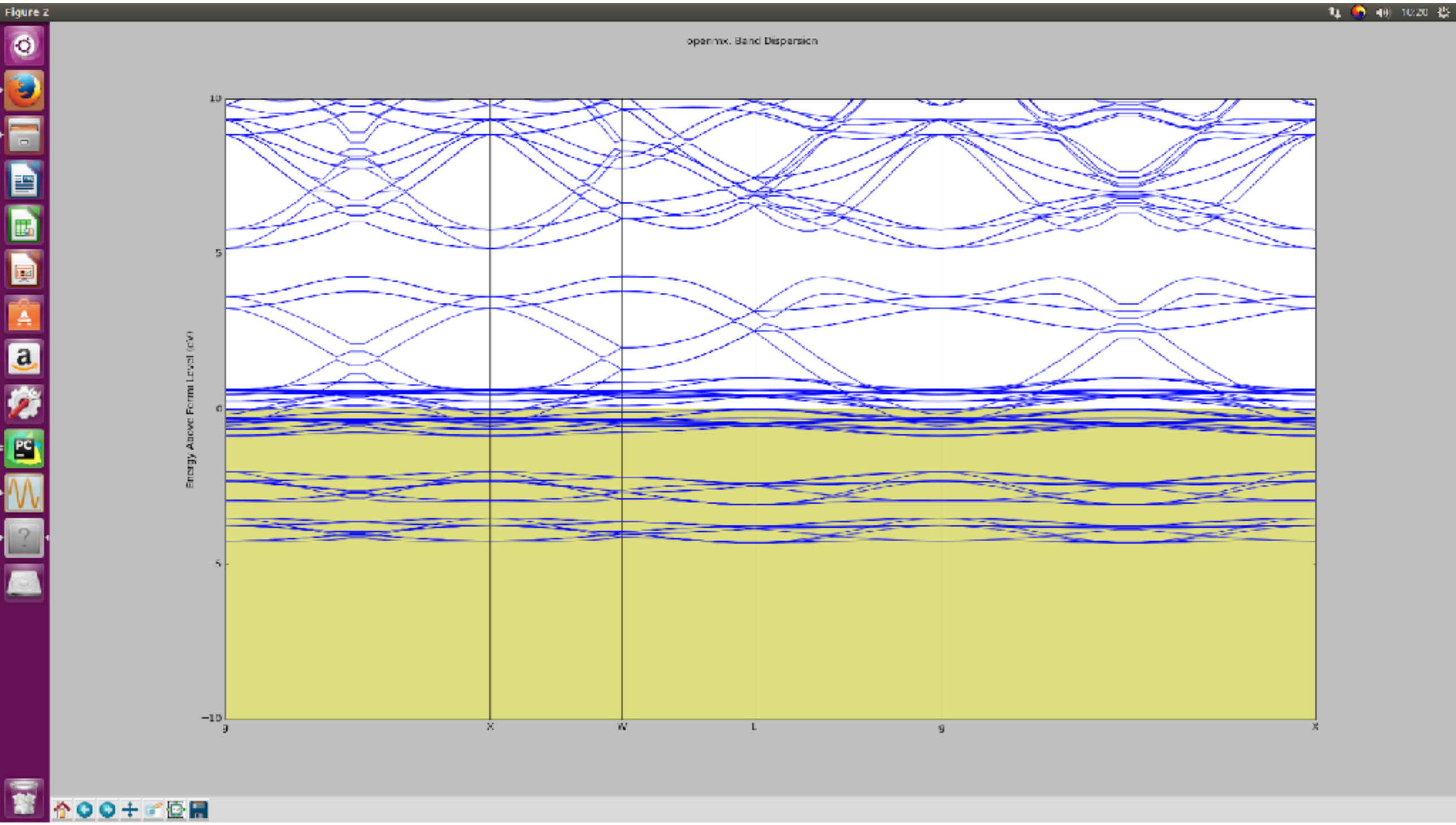


Figure 2
10:20
opermx. Band Dispersion
Energy Above Fermi Level (eV)
10
5
0
−5
−10
Γ
X
W
L
Γ
X

Molecular Orbitals

- To calculate molecular orbitals, specify at least either homos or lumos as a positive integer. If you want to visualise the orbitals, you need to have a VESTA* executable located at `absolute_path_of_vesta`.

- If `eigenvalue_solver='Cluster'`, `kpts` can only have one point.

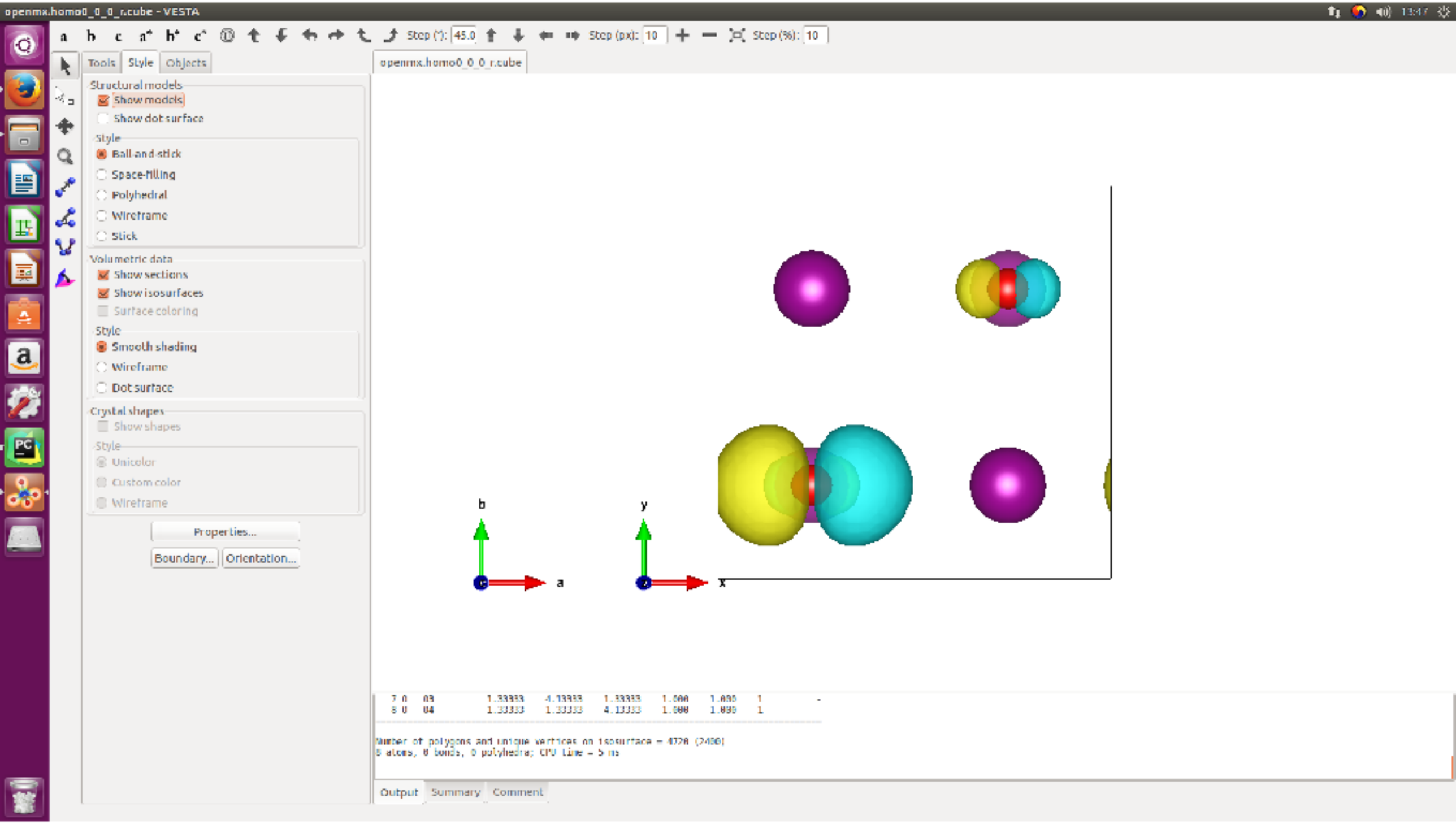
```
kpts = [(0, 0, 0), (1, 0, 0), (1, 1, 0), (1, 1, 1)]

calc = OpenMX(homos=3, lumos=3, mo_kpts=kpts,
absolute_path_of_vesta=HOME_DIRECTORY/src/VESTA-x86_64/VESTA)
```

- To visualise the orbitals:

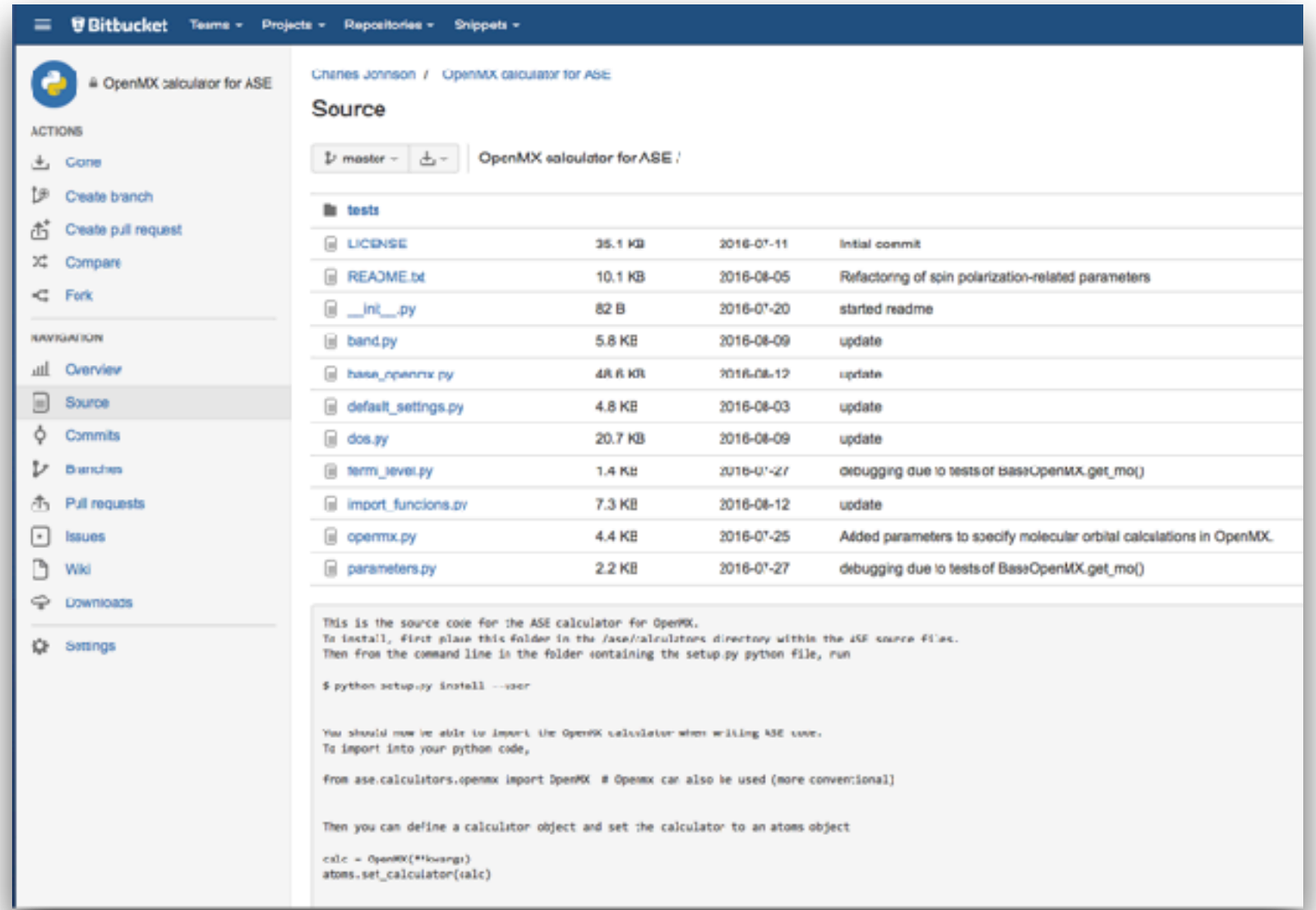
```
calc.get_mo(real=True, imaginary=False)
```

*see http://jp-minerals.org/vesta/ for information about this 3D visualization program for structural models.


openmx.homo0_0_0_r.cube - VESTA
Step (°): 45.0
Step (px): 10
Step (%): 10
Tools
Style
Objects
openmx.homo0_0_0_r.cube
Structural models
Show models
Show dot surface
Style
Ball and stick
Space-filling
Polyhedral
Wireframe
Stick
Volumetric data
Show sections
Show isosurfaces
Surface coloring
Style
Smooth shading
Wireframe
Dot surface
Crystal shapes
Show shapes
Style
Unicolor
Custom color
Wireframe
Properties...
Boundary...
Orientation...
b
a
y
x
Number of polygons and unique vertices on isosurface = 4720 (2400)
5 atoms, 0 bonds, 0 polyhedra; CPU time = 5 ms
Output
Summary
Comment

Developments

https://bitbucket.org/

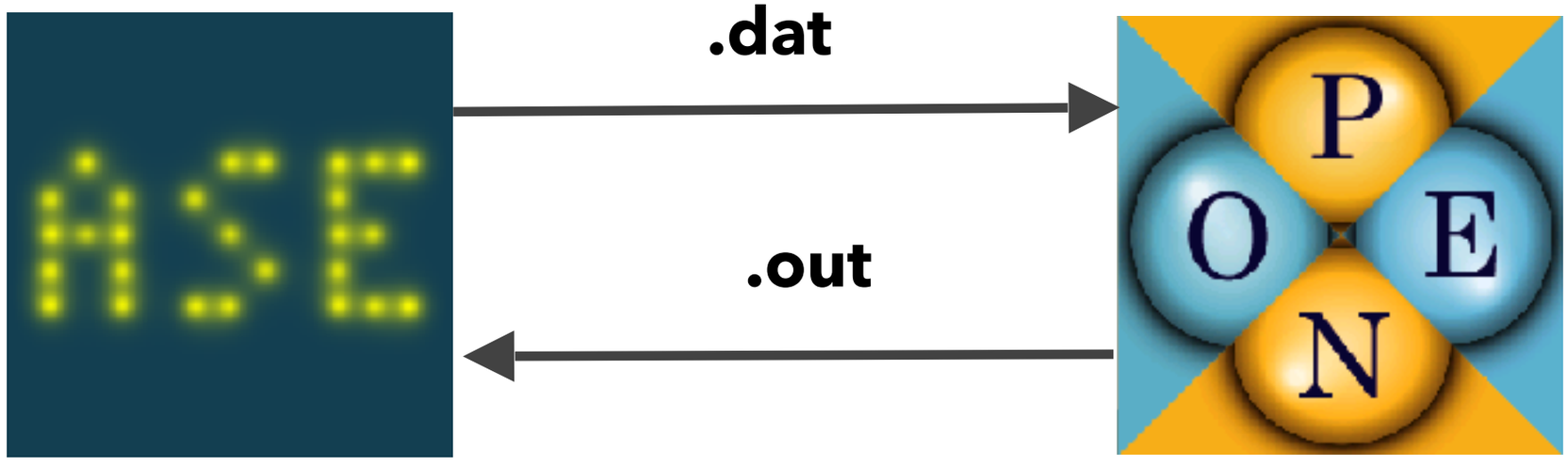


If you like to join, please send a message to Prof. Jaejun Yu at <jyu@snu.ac.kr>.

Prospects

- Tools:
  - Make an easier learning curve
  - Interactive monitoring SCF and MD loops
  - Importing DB like ICSD and cross-checking
  - Proper choice of PAO's and pseudo-potentials

- Applications:
  - High-throughput screening calculations
    ➔ materials design
  - Automated Interactive control of calculations
    ➔ in conjunction with other tools like AiiDA



Questions or Comments?

Ref.:

https://wiki.fysik.dtu.dk/ase/ - ASE

http://www.openmx-square.org/ - OpenMX