Atomic simulation environment

EUSpec
Modern tools for Spectroscopy on Advanced Materials
January 18, 2018

Jens Jørgen Mortensen
Department of Physics
Technical University of Denmark

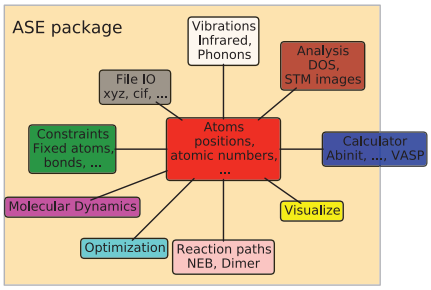
- Python
- ASE
- Excercises

Plan

- Quick ASE introduction
- Introduction to Python
  - What is Python?
  - Datatypes
  - NumPy
- Overview of ASE (atomic simulation environment)
  - The `Atoms` object
  - Building stuff (molecules, surfaces, ...)
  - Doing stuff with atoms (MD, optimization, ...)
  - Database module
  - Command-line tools
  - Infrastructure

What is ASE?

ASE is a Python library for working with atoms:



`http://wiki.fysik.dtu.dk/ase`

ASE-papers

- **An object-oriented scripting interface to a legacy electronic structure code**
  Sune R. Bahn and Karsten W. Jacobsen
  Comput. Sci. Eng., Vol. 4, 56-66, 2002
- **The Atomic Simulation Environment — A Python library for working with atoms**
  Ask Hjorth Larsen, Jens Jørgen Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dulak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. B. Jensen, K. Kaasbjerg, J. Kermode, J. R. Kitchin, E. L. Kolsbjerg, J. Kubal, S. Lysgaard, J. B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and K. W. Jacobsen
  2017 J. Phys.: Condens. Matter 29 273002

A recipe

```
from ase import Atoms
from ase.calculators.nwchem import NWChem
from ase.optimize import BFGS
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)
write('H2.xyz', h2)
e = h2.get_potential_energy()
print(e)
```

Result

```
$ python3 script.pt
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
-31.492847800329216
```

What is Python?

`http://www.python.org`

- Interpreted language
- Easy to read and write, and very expressive
- Great for scripting a calculation
- Great for small and large programs (Instagram, Youtube, GPAW, ...)
- Optimized for programmer productivity
- Can be extended in C/C++/Fortran
- General purpose language
- Open Source

The ecosystem around Python

Numpy: Base N-dimensional array package

Scipy: Fundamental library for scientific computing (interpolation, integration, FFT, linear algebra, optimization, sparse matrices and much more)

Matplotlib: Comprehensive Plotting

Sympy: Symbolic mathematics.

Jupyter: Interactive data science

Python 3

We'll be using Python 3, so add this to your `.bashrc` or similar:

```
alias p=python3
```

Python 3: Many things fixed in the language:

- Better unicode handling
- `print()` is now a function
- No more `1 / 2 == 0`
- ...

There is no reason to use Python 2.7 anymore

- released on July 3, 2010
- supported until April 2020

Getting help

- Google: `https://stackoverflow.com`
- `https://python.org`
  - Documentation
  - Tutorials
  - Standard library modules
- The Hitchhiker's Guide to Python: `http://docs.python-guide.org`
  - Best practices
  - Text editors: emacs, vi, pycharm, idle, jupyter, ...
- Interactive interpreter:
  - `help()`, `dir()` or tab-complete
- The source code

How to run Python code?

Use the Python interpreter directly:

```
$ python3 script.py
...
$ sbatch ... script.py
...
$ python3
>>> 2 + 2
4
>>> ^D
$
```

or through a Jupyter notebook running in your browser.

Datatypes

```
None                # NoneType (missing)
True, False         # bool
1                   # int
1.2                 # float
2.0 + 3.0j          # complex
{1: 2, 2: 'abc'}    # dics
{1, 2, 3}           # set
[1, 2, 3]           # list
(1, 2, 'three')     # tuple
'Hello'             # str
```

Mutable: `list`, `set`, `dict` (can't be used as keys in a `dict`)
More things: `def`, `zip()`, `enumerate()`, `sorted()`, `round()`, `reversed()`, `min()`, `max()`, `abs()`, `len()`, `int()`, `float()`, `divmod()`, `any()`, `all()`.

Strings

```
>>> age = 2018 - 1991
>>> 'Age: {} years.'.format(age)
'Age: 27 years.'
>>> x = 2**0.5
>>> 'x = {:.2f}'.format(x)
'x = 1.41'
# or
>>> 'x = {x:.2f}'.format(x=x)
'x = 1.41'
```

New Python 3.6 syntax (f-strings):

```
>>> f'Age: {age} years.'
'Age: 27 years.'
>>> f'x = {x:.2f}'
'x = 1.41'
```

Importing stuff

```
import time
time.sleep(1)

from time import sleep
sleep(1)

from time import sleep as zzz
zzz(1)

from time import *
sleep(1)
```

Some conventions:

```
import matplotlib.pyplot as plt
import numpy as np
```

stdlib

The best stuff:

- collections: `defaultdict`, `Counter`, `namedtuple`
- os.path (deal with file-system paths)
- pathlib (modern way to deal with file-system paths)
- subprocess: Run shell commands
- pickle, json, csv: Write stuff to files
- argparse: Create command line interfaces
- re: Regular Expressions
- math (might as well use numpy): `sin`, `pi`, ...
- functools: `partial`

Others: `multiprocessing`, `itertools`, `time`, `this`, ...

NumPy

```
>>> a = [1, 2, 3]
>>> a * 2
[1, 2, 3, 1, 2, 3]
a - a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -:
    'list' and 'list'
```

`http://docs.scipy.org/doc/numpy/reference/`

```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
>>> a * 2
array([2, 4, 6])
>>> a.dtype  # size, shape, ndim
dtype('int64')
```

NumPy ...

```
>>> m1 = np.array([[1, 0],
...                [0, -1]])
>>> m2 = np.ones((2, 2))
>>> m1
array([[ 1,  0],
       [ 0, -1]])
>>> m2
array([[ 1.,  1.],
       [ 1.,  1.]])
>>> m1 * m2
array([[ 1.,  0.],
       [ 0., -1.]])
>>> m1 @ m2
array([[ 1.,  1.],
       [-1., -1.]])
```

Bad style

Can you find the 6 "errors"?

```
def is_animal(x):
  all_animals = ['cow', 'cat', 'dog']
  return x in all_animals:

stuff = ['cow', 'car', 'cat']

animals = []
for i in range(len(stuff)):
  if (is_animal(stuff[i]) == True):
    animals.append(x)
```

Good style

```
all_animals = {'cow', 'cat', 'dog'}

def is_animal(x):
    return x in all_animals:

stuff = ['cow', 'car', 'cat']

animals = [x for x in stuff
           if is_animal(x)]
```

File IO

```
a = np.array([...])
np.savetxt('data.txt', a)
a = np.loadtxt('data.txt')

dct = {'a': 1.2,
       'b': [1, 2, 3]}
import json
with open('data.json', 'w') as f:
    json.dump(dct, f)

x = [...]
y = [...]
with open('xy.txt', 'w') as f:
    for a, b in zip(x, y):
        print(a, b, file=f)
```

There is also the `pickle` module, but ...

Exercise 1

We want something like this:

```
$ cat input.txt
car 100.0  1
cat   0.2  2
cow   3.2 10
$ python3 cut1.py input.txt 2 3
100.0 1
0.2 2
3.2 10
```

Quick'n'dirty solution

```
from sys import argv
f = open(argv[1])
columns = [int(arg) - 1
           for arg in argv[2:]]
for line in f:
    parts = line.split()
    print(' '.join(parts[i]
                   for i in columns))
```

Exercise 2

Modify the script so that it can calculate the sum of a column:

```
$ python3 cut2.py input.txt 2
103.4
```

Hint:

```
>>> float('100.0')
100.0
```

See next pages for a nicer solution with all the bells and whistles (uses `argparse`).

A nicer solution

```
import argparse
parser = argparse.ArgumentParser(
    description='Cut out selected columns')
parser.add_argument(
    '-c', '--columns',
    help='Columns to select. '
    'Defaults to first column only.')
parser.add_argument(
    '-s', '--sum', type=int, metavar='N',
    help='Sum column number N.')
parser.add_argument('file',
                    help='Input file.')

args = parser.parse_args()
```

...

A nicer solution (part 2)

...

```
if args.columns:
    columns = [int(c) - 1
               for c in
               args.columns.split(',')]
else:
    columns = []
```

...

A nicer solution (part 3)

...

```
total = 0.0
with open(args.file) as f:
    for line in f:
        parts = line.split()
        if columns:
            print(' '.join(parts[i]
                           for i in columns))
        if args.sum:
            x = float(parts[args.sum - 1])
            total += x
if args.sum:
    print(f'Total: {total:.3f}')
```

Installing ASE from PyPI

Get your code from PyPI instead of your OS.

```
$ pip3 install ase --user
$ pip3 install jupyter --user
```

(you may need to add `~/.local/bin/` to your `$PATH`)

If you want an isolated installation:

```
$ python3 -m venv euspec
$ source euspec/bin/activate
(euspec) $ pip install ase
(euspec) $ ...
```

Does it work?

```
$ python3
>>> import ase
>>> ase
<module 'ase' from '/.../__init__.py'>
>>> ^D
$ ase
$ ase info
$ ase test
$ ase build H2O water.xyz
$ ase gui water.xyz
$ ase completion >> ~/.bashrc
```

A tour of the web-page

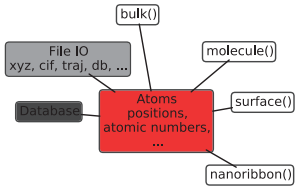
https://wiki.fysik.dtu.dk/ase/

The Atoms object

What's inside?

- Positions, atomic numbers, velocities, masses, initial magnetic moments, charges, tags
- Unit cell
- Boundary conditions
- (Calculator)
- (Constraints)

Building the Atoms object



File IO:

- Standard file formats: .xyz, .cif, .cube, ... (curently 72 different ones)
- ASE's own formats: .traj, .json, .db

Building the Atoms object ...

```
from ase import Atoms
h2 = Atoms('H2', positions=[(0, 0, 0),
                            (0, 0, 0.74)])
from ase.build import molecule, bulk
water = molecule('H2O')
si2 = bulk('Si', 'diamond', a=5.4)
from ase.spacegroup import crystal
a, c, u = 4.584, 2.953, 0.3
rutile = crystal(['Ti', 'O'],
                 basis=[(0, 0, 0), (u, u, 0)],
                 spacegroup=136,
                 cellpar=[a, a, c,
                          90, 90, 90])
from ase.build import fcc110
slab = fcc110('Pt', (2, 1, 7), a=4.0,
              vacuum=6.0)
from ase.build import add_adsorbate
```

Building the Atoms object, continued ...

```
from ase.build import surface
nasty_cut = surface(rutile, (1, 1, 0),
                    layers=5)
from ase.io import read
q = read('quarts.cif')
import ase.db
con = ase.db.connect('mystuff.db')
atoms = con.get_atoms(xc='LDA', H=0)
# same as this:
atoms = read('mystuff.db@xc=LDA,H=0')
```

Methods and attributes of the Atoms object

```
a.pop()  # remove last atom
a.append('Xe')  # append xenon atom
a.extend(Atoms('H2'))  # append 2 hydrogens
a[1]  # second atom
del a[-3:]  # delete three last atoms
```

Other methods: `repeat()`, `translate()`, `center()`, `rotate()`, `get_distance()`, `get_angle()`, `get_moments_of_inertia()`, `get_potential_energy()`, `get_forces()`, `get_stress()`, ...
Special attributes: `positions`, `numbers`, `cell`, `pbc` (Numpy n-d arrays)

```
x1, x2 = a.positions[-2:, 0]  # x-coords of
                              # last two atoms
a.positions[-2:, 0] += 0.1  # translate atoms
a.positions[:, 2].max()  # maximum z-coord
```

Exercice 3

Build a water molecule and write an *xyz* file.

Help:

- *Bond length: 0.97 Å*
- *Angle: 104 degrees*
- *Use `ase.Atoms` for creating it*
- *Use `ase.io.write()` for writing*
- *Use `sin()` and `cos()` from `math` or `numpy`*
- *Alternatively, use the `set_distance()` and `set_angle()` methods of the `Atoms` object*

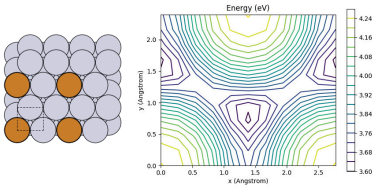
A simple Python program

How to use ASE functions, for-loops and string formatting:

```
from ase.build import fcc111
from gpaw import GPAW, PW
a = 3.6
k = 10
for n in range(1, 5):
    slab = fcc111('Cu', size=(1, 1, n),
                  a=a, vacuum=5.0)
    slab.calc = GPAW(mode=PW(ecut=400),
                     kpts=(k, k, 1),
                     txt=f'Cu{n}.txt')
    slab.get_potential_energy()
```

Run with: `python3 script.py`

PES of a Cu atom on a Pt(111) surface



ASE's calculators

DFT, *ab initio*:

Abinit, Castep, CP2K, Crystal14, deMon, DMol, ELK, Exciting, Fleur, FHI-Aims, Fleur, Gaussian, GPAW, GULP, Jacapo, JDFTx, NWChem, Octopus, ONETEP, QuantumEspresso, SIESTA, Turbomole, VASP

(Semi-)empirical potentials:

Amber, ASAP, Atomistica, DFTB+, EAM, EMT, Gromacs, Hotbit, LAMMPS, Lennard-Jones, MOPAC, Morse, OpenKIM

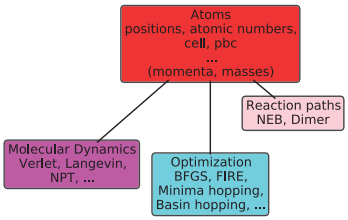
vdW-corrections:

Tkatchenko-Scheffler, Grimme-D3

Three types of calculators

1. - ASE writes input file
   - ASE starts Fortran/C/C++ code (maybe in parallel)
   - ASE waits for Fortran/C/C++ code to finish
   - ASE reads output file
2. Same as above, except that the Fortran code doesn't stop - it waits for a new input file with new coordinates. Communication via sockets or stdin/stdout (or some other mechanism).
3. Python solution (EMT, EAM, TIPnP, ...). For parallel calculations, the Python interpreter runs on all processes (GPAW, ASAP).

Optimization, MD and minimum energy paths



ASE sub-modules:

collections, constraint, build, data, db, dft, dimer, eos, ga, geometry, infrared, io, md, neb, neighborlist, optimize, phasediagram, phonons, spacegroup, thermochemistry, transport, units, vibrations, visualize.

Database module in ASE

In each row we have:

Taxonomy:

- `Atoms` object (positions, atomic numbers, ...)
- ID, user-name, creation and modified time
- Constraints
- Calculator name and parameters
- Energy, forces, stress tensor, dipole moment, magnetic moments

Folksonomy:

- Key-value pairs (string or number)

Additional stuff:

- Extra data (band structure, dos, ...)

`https://wiki.fysik.dtu.dk/ase/ase/db/db.html`

ase.db examples

```
from ase.db import connect
from ase import Atoms
con = connect('abc.db')
h = Atoms('H')
con.write(h)
from ase.calculators.emt import EMT
h.calc = EMT()
h.get_forces()
con.write(h, abc=42)
```

```
$ ase db abc.db --columns +abc
id|age|formula|calculator|energy| fmax|pbc|abc
 1|15m|H      |          |      |     |FFF|
 2|15m|H      |emt       | 3.210|0.000|FFF| 42
Rows: 2
```

ase.db examples ...

```
$ ase db abc.db abc=42 --json
{"1": {
 "calculator": "emt",
 "calculator_parameters": "{}",
 "cell": [[1, 0, 0], [0, 1, 0], [0, 0, 1]],
 "ctime": 16.422649618451555,
 "energy": 3.21,
 "forces": [[0.0, 0.0, 0.0]],
 "key_value_pairs": {"abc": 42},
 "mtime": 16.422670641858346,
 "numbers": [1],
 "pbc": [false, false, false],
 "positions": [[0.0, 0.0, 0.0]],
 "user": "jensj"},
"ids": [1], "nextid": 2}
```

`https://cmr.fysik.dtu.dk/`

ASE command-line tools

```
$ ase <command> [options]
```

Commands:

help	Help for sub-command
info	Print information about files or system
test	Test ASE
gui	ASE's graphical user interface
db	Manipulate and query ASE database
run	Run calculation with one of ASE's calculators
band-structure	Plot band-structure
build	Build an atom, molecule or bulk structure
...	

ASE command-line tools (continued)

...	
eos	Calculate equation of state
ulm	Manipulate/show content of ulm-file
find	Find files with atoms in them
nomad-upload	Upload files to NOMAD
convert	Convert between file formats
reciprocal	Show the reciprocal space
completion	Add tab-completion for Bash

Use `ase xxx --help` for help.

CLI examples

```
$ ase build CO | ase run nwchem -f 0.05
Running: CO
LBFGS:    0  08:36:52  -3041.669444     0.8139
LBFGS:    1  08:36:52  -3041.659936     1.5638
LBFGS:    2  08:36:53  -3041.672467     0.0457
$ ase gui CO.traj@-1
```

```
$ ase build -x fcc Cu -a 3.6
$ ase run emt Cu.json --equation-of-state=5,2
Running: Cu
$ ase eos Cu.traj
# filename points  volume energy bulk modulus
#                [Ang^3]   [eV]        [GPa]
Cu.traj         5  11.565 -0.007      134.419
```

Slicing (@start:stop:step)

For files containing more than one atomic configuration, you can append '@start:stop:step' to the filename in order to do slicing:

```
from ase.io import read
a = read('relaxation.traj@start:stop:step')
```

- '@:': all
- '@-1': last
- '@0': first
- '@-2:': last two
- '@:3': first three
- '@::3': every third
- '@abc=42,Cu>2': database query

Infrastructure

- Git repository: https://gitlab.com/ase/ase
  Code, merge-requests and issues.
  Read this: https://wiki.fysik.dtu.dk/ase/development/contribute.html
- Web-page: https://wiki.fysik.dtu.dk/ase/
- PyPI: https://pypi.org/project/ase/
- Tar-files can be downloaded from PyPI
- Monthly code-sprints in Lyngby 10:00-15:00 (first Tuesday every month)
- Mail list: ase-users@listserv.fysik.dtu.dk
- IRC channel on freenode.net (#ase).
- Quality Assurance Department: you!

More exercises

- Diffusion of gold atom on Al(100) surface (constraint)
- Self-diffusion on the Al(110) surface (NEB + Dimer method exercise)
- Diffusion of gold atom on Al(100) surface
- Nudged elastic band (NEB) calculations
- Equation of state calculations
- Finding lattice constants of HCP nickel
- Atomization energy of nitrogen
- Try your favourite electronic structure code ...

Links

ASE:

- http://wiki.fysik.dtu.dk/ase
- http://wiki.fysik.dtu.dk/ase/dev
- https://gitlab.com/ase/ase
- http://cmr.fysik.dtu.dk
- https://doi.org/10.1088/1361-648X/aa680e
- http://psi-k.net/download/highlights/Highlight_134.pdf

Python:

- http://www.python.org/
- http://docs.scipy.org/doc/numpy/reference/
- http://docs.scipy.org/doc/scipy/reference/
- http://matplotlib.org/
- http://jupyter.org/
- http://docs.python-guide.org

Finally ...

Thank you for your attention
Questions?