Morten N. Gjerding & CAMD mogje@dtu.dk
Department of Physics, Technical University of Denmark
ASE Workshop, Gothenburg, Nov. 2019

ATOMIC SIMULATION RECIPES

Proposal

We should have some standardized form for scripts.

Outline

- What is the problem
- Our suggestion
  - How should it work in practice?
  - Examples
  - Design choices

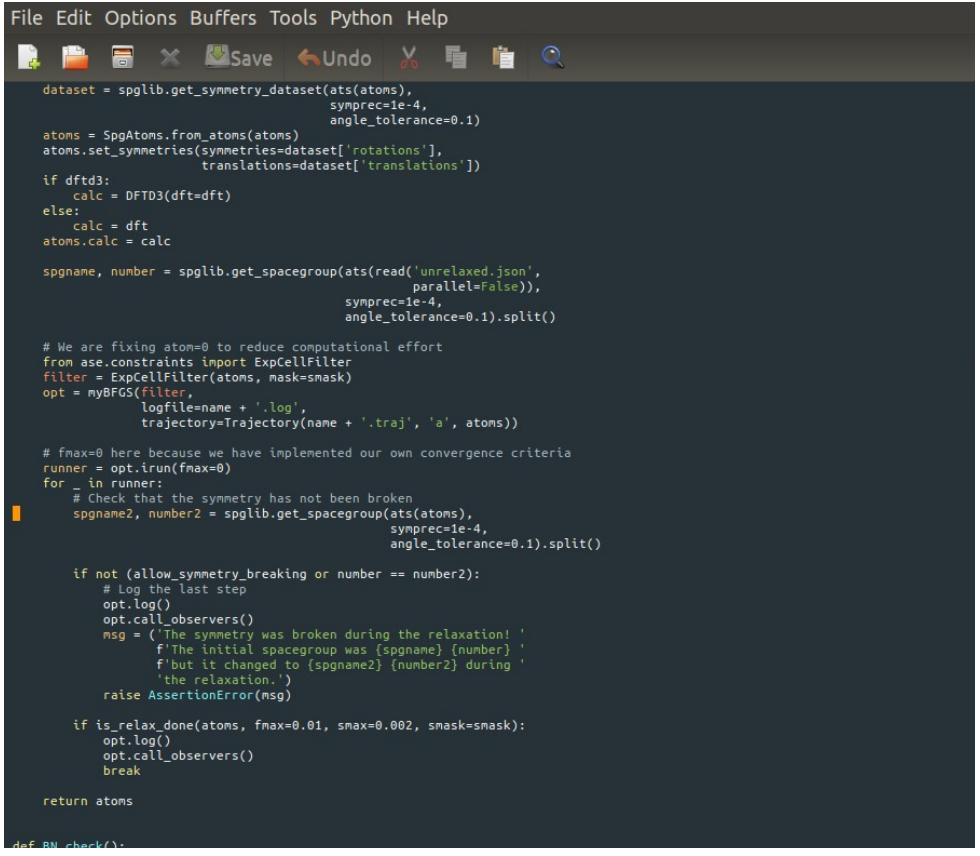
Normal ASE workflow

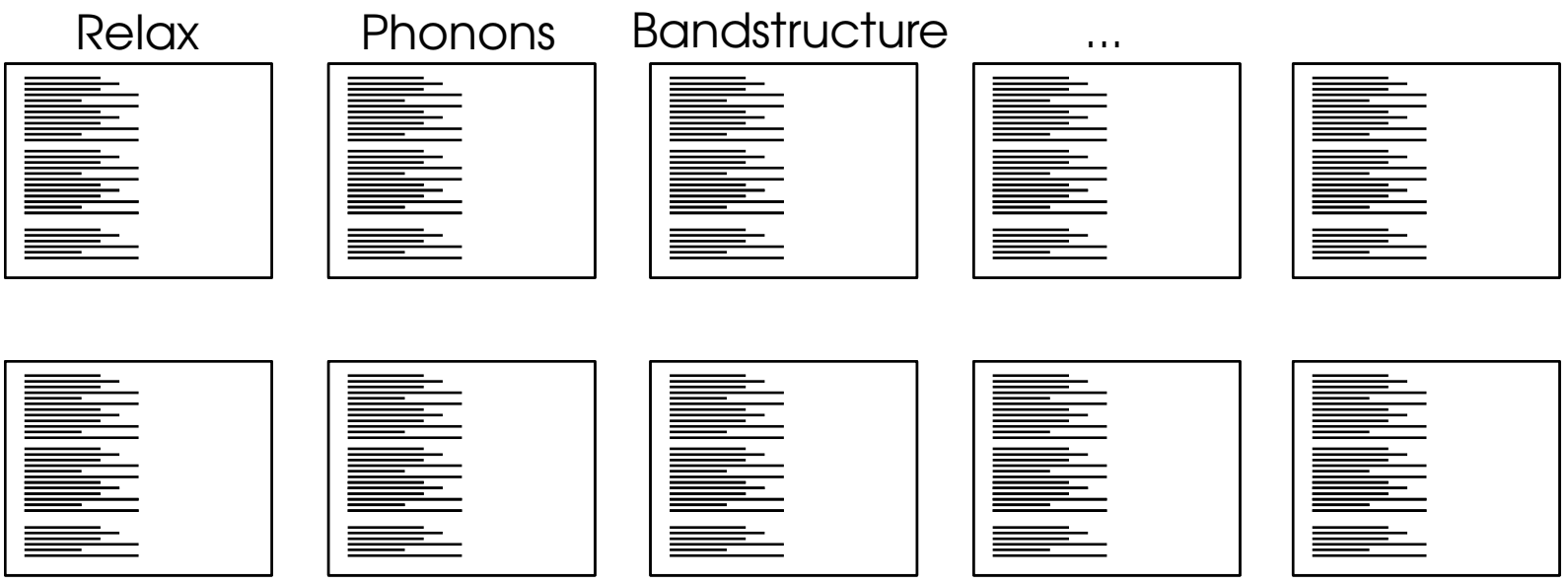
To relax a structure I would probably start with a script like this.

I would gradually expand my script:

- Save each relaxation step
- Relax cell: `UnitCellFilter()`
- Etc…

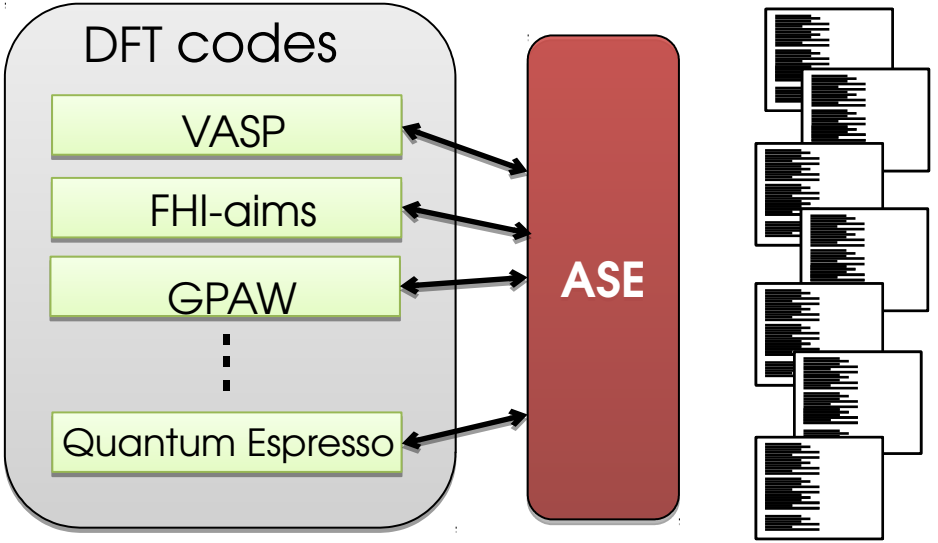
In time the script would have grown significantly.





These scripts become my recipes to calculate properties

The big picture



"Pile of scripts"
- Adam Jackson

Waste of student time?

- Many (most, all?) students go through this process
  - Writing a script, improving on it
  - Doing the same mistakes
  - Implementing variants the same good ideas
  - Converge to (mostly) the same script
- Learning experience?
  - Some important lessons in this process
  - Some not so important
  - Some definitely a waste of time

Using new modules

Developer

Module

Student

Script

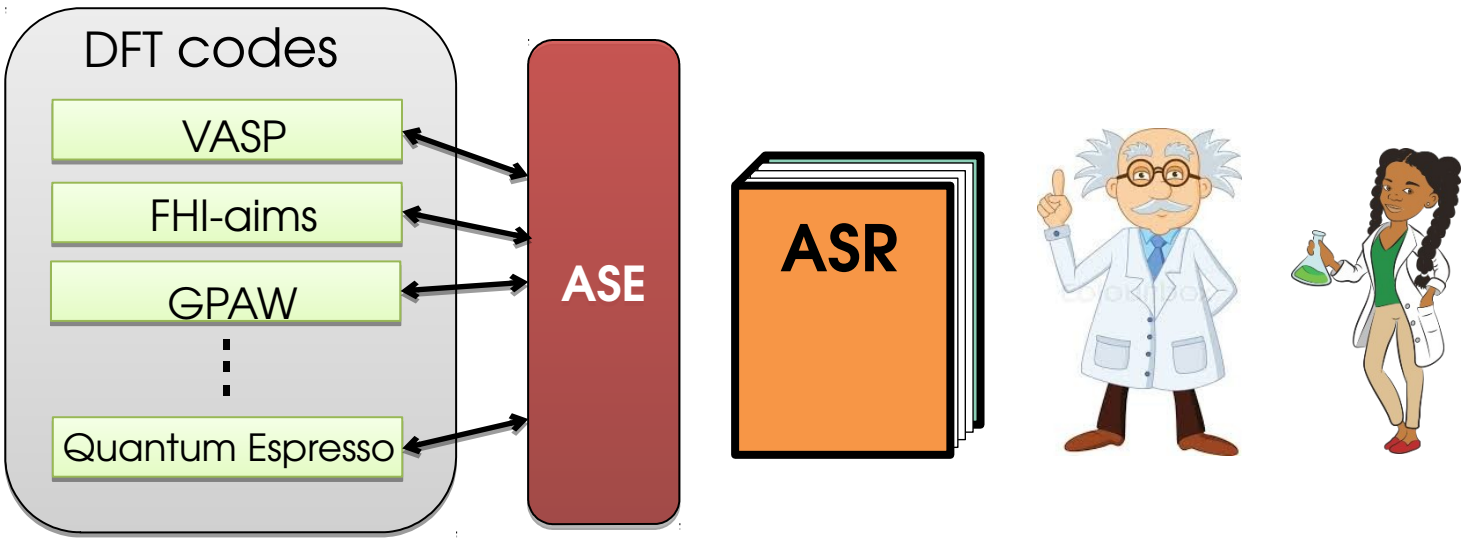
Tutorials are often solving an artificial problem, scripts solve a real world problem.

Do we need a place for those?

Anecdote

- AIMS workshop, ASE tutorial Barcelona 2019
- “Are there any predefined scripts for relaxing a structure?”
- Do we need a repository and some standardized form for scripts? Could we even agree on something?

The big picture



What should such a package be/do/not do?

OUR SUGGESTION
(BASED ON OUR NEEDS,
UP FOR DISCUSSION)

What is a recipe?

- **A annotated function that returns its results as a dictionary.**
- Results are saved to a json file.
- The `@command` decorator takes care of everything.
- In particular, gives the function a command line interface.
- The ASR package is just a collection of scripts containing recipes.

```
File Edit View Search Terminal Help
File Edit Options Buffers Tools Python Virtual Envs Elp
from asr.core import command, option


@command()
@option('--fmax', help='Minimum force tolerance.')
def relax(fmax=0.01):
    """This function relaxes a structure"""
    from ase.optimize.bfgs import BFGS
    BFGS(...).run(fmax)
    results = {'energy': ...}
    return results


if __name__ == '__main__':
    relax.cli()
```

Command line interface

```
$ python3 -m asr.relax --help
Usage: python -m asr.relax [OPTIONS]

  This function relaxes a structure

Options:
  --fmax FLOAT  Minimum force tolerance.  [default: 0.01]
  -h, --help    Show this message and exit.
```

Full disclosure:

The actual relax recipe

```
@command('asr.relax',
         requires=['unrelaxed.json'],
         creates=['structure.json'])
@option('-c', '--calculator', help='Calculator and its parameters.')
@option('--d3/--nod3', help='Relax with vdW D3')
@option('--fixcell', is_flag=True, help='Don\'t relax stresses')
@option('--allow-symmetry-breaking', is_flag=True,
        help='Allow symmetries to be broken during relaxation')
def main(calculator={'name': 'gpaw',
                     'mode': {'name': 'pw', 'ecut': 800},
                     'xc': 'PBE',
                     'kpts': {'density': 6.0, 'gamma': True},
                     'basis': 'dzp',
                     'symmetry': {'symmorphic': False},
                     'convergence': {'forces': 1e-4},
                     'txt': 'relax.txt'
```

Example: Relax silver

```
$ ase build Ag unrelaxed.json --crystal-structure fcc

$ python3 -m asr.relax --calculator "{'name':'emt'}"
Running asr.relax(calculator={'name': 'emt'}, d3=False,
fixcell=False, allow_symmetry_breaking=False)

$ ls
relax.log  relax.traj  results-asr.relax.json
structure.json  unrelaxed.json
```

Result files

```
$ cat results-asr.relax.json
{
 "energy": -0.00036639685640693 64,
 "__params__": {
  "calculator": {"name": "emt"},
  "d3": false,
  "fixcell": false,
  "allow_symmetry_breaking": false
 },
 "__versions__": {
  "asr": "19.8.20-8ad12500add3f21047dc2b5a91cecbd3daa07d39",
  "ase": "3.19.0b1-da3288338c6896a9a5b15322d303471d826f7486",
 }
}
```

Result files

```
$ cat results-asr.relax.json
{
 "energy": -0.00036639685640693 64,
…


"__creates__": {
  "structure.json": "1bb6cf42c49864c81d7f3a394918b71a"
 },
"__requires__": {
  "unrelaxed.json": "be8cc1a545f45522e9880635bfe955c5"
 },

}
```

Important: Direct relation between scripts and CLI

```
$ python3 -m asr.relax --calculator "{'name':'emt'}"
Running asr.relax(calculator={'name': 'emt'}, d3=False,
fixcell=False, allow_symmetry_breaking=False)
```

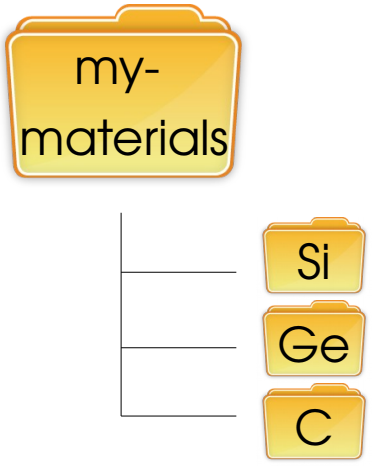
=

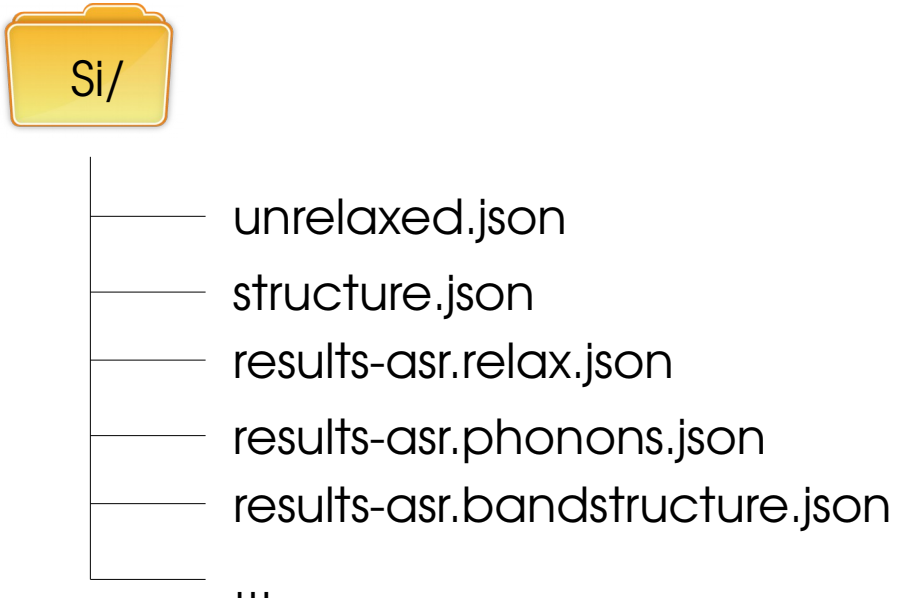
```python
from asr.relax import main as relax

relax(calculator={'name': 'emt'}, d3=False,
      fixcell=False, allow_symmetry_breaking=False)
```

Parameter files

```
$ p -m asr.setup.params asr.relax:calculator "{'name':'emt'}"

$ cat params.json
{
 "asr.relax": {
  "calculator": {
   "name": "emt"
  }
 }
}
```

One folder contains one material





Collecting to a database becomes easy

Recipes so far

Recipes that would work for all calculators

- `asr.relax`
- `(asr.phonons)`
- `asr.stiffness`
- `asr.structureinfo`
- `(asr.convex_hull)`
- `((asr.bandstructure))`

Recipes that work only for GPAW

- `asr.gs@calculate`
- `asr.bandstructure@calculate`
- `asr.berry`
- `asr.polarizability`
- `asr.effective_masses`
- `asr.fermi_surface`
- `...`

Summary of design choices

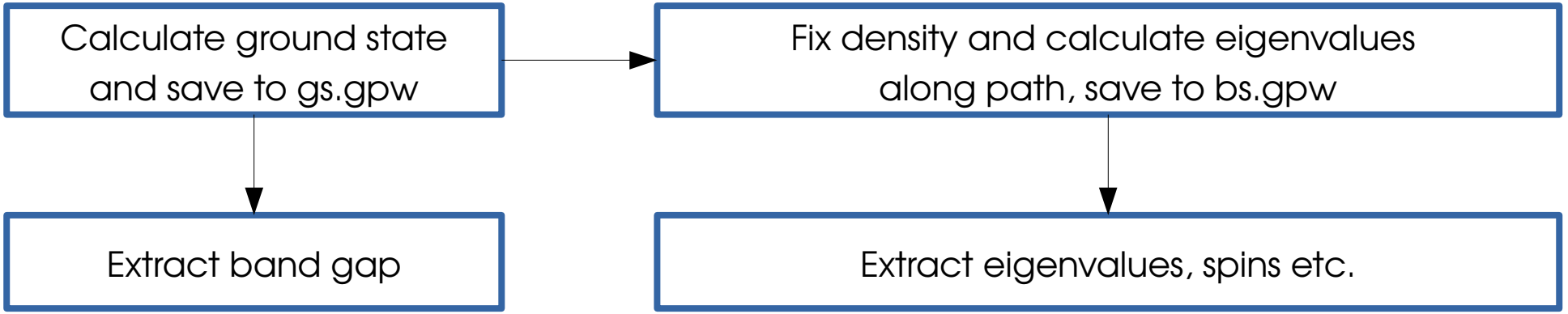
- Recipe = Decorated, annotated Python function
- ASR = A collection of recipes
- Results saved in json files
  - Contains parameters, versions, file hashes
- One-to-one correspondence between CLI and scripts.
- One folder = one material, only have to run a recipe in a folder once.
- Put parameters in a parameter file.
- ASR is available on PyPI
  - `$ pip install asr`

Discussion

- Should we have some standardized script format?
- Data provenance: "*Data provenance provides a historical record of the data and its origins.*"
  - Checksums, git-hashes = reasonably data provenant?
- Issues: design choices, generality, file formats, documentation.
- ASR is available on PyPI
  - `$ pip install asr`
- Thank you for your attention.

Advanced properties

- Many properties are difficult to generalize to other calculators because they depend on the detailed workings of each calculator
- Many properties constitute small workflows in themselves that might be different for every calculator.
- For example, to calculate a bandstructure in GPAW



Workflow

```
$ asr run asr.gs@calculate
$ asr run asr.bandstructure@calculate
$ asr run asr.bandstructure

# I also want the band gap

$ asr run asr.gs
```

Dependencies

asr.gs

```python
from asr.core import command


@command()
def calculate():
    """Calculate ground state and save to gs.gpw"""
    ...


@command(dependencies=['asr.gs@calculate'])
def main():
    """Extract data (band gap etc.) from gs.gpw"""
    ...


if __name__ == '__main__':
    main.cli()
```

asr.bandstructure

```python
from asr.core import command


@command(dependencies=['asr.gs'])
def calculate():
    """Fix density and save raw data in bs.gpw"""
    ...


@command(dependencies=['asr.bandstructure@calculate'])
def main():
    """Extract eigenvalues and save bandstructure"""
    ...


if __name__ == '__main__':
    main.cli()
```

```
$ python3 -m asr.bandstructure
$ python3 -m asr.gs
```