Density Functional Theory (DFT) calculations in PySCF

This set of exercises comprises all the information you need to run a DFT calculation in PySCF. After giving a detailed account of how to install the software, you will learn how to create the inputs and apply the DFT knowledge to your own electronic structure calculation.

Installation of PySCF

Below you can find instructions on how to install PySCF for non-developers, adapted from the PySCF website (`https://pyscf.org/index.html`).

Please connect to meluxina and move to the `/project/home/p201028` directory:

```
cd /project/home/p201028
```

Then, request an interactive job on a CPU node for 1 hour:

```
salloc -A p201028 --reservation=cpudev -q dev -N 1 -t 1:0:0
```

Install PySCF using `pip`:

```
pip install --prefer-binary pyscf
```

The `pip` package provides a precompiled PySCF code. For more options on how to install PySCF, please refer to the website: `https://pyscf.org/user/install.html`

Inputs

To perform a DFT calculation, two main inputs are required: the **molecular geometry (or crystal structure)** and the choice of **basis sets**.

Initializing a molecule

There are three ways to define a molecule.

The first one is to use the `Mole.build()` method to initialize a molecule, in this case H_2O:

```
from pyscf import gto
mol = gto.Mole()
mol.build(
    atom = '''O 0 0 0; H  0 1 0; H 0 0 1''',
    basis = '''sto-3g''')
```

To confirm that the molecular geometry and basis sets have been set up properly, you can run:

```
print(mol.atom)
print(mol.basis)
```

The second one is to assign the geometry, basis, to the `Mole` object, followed by calling the `build()` method:

```
from pyscf import gto
mol = gto.Mole()
mol.atom = 'O 0 0 0; H  0 1 0; H 0 0 1'
mol.basis = 'sto-3g'
mol.build()
```

In this case as well, you can confirm that the molecular geometry and basis sets have been set up correctly by running:

```
print(mol.atom)
print(mol.basis)
```

The third way is to use the shortcut functions `pyscf.M()` or `Mole.M()`. These functions pass all the arguments to the `build()` method:

```
import pyscf
mol = pyscf.M(
    atom = '''O 0 0 0; H  0 1 0; H 0 0 1''',
    basis = '''sto-3g''')

from pyscf import gto
mol = gto.M(
    atom = '''O 0 0 0; H  0 1 0; H 0 0 1''',
    basis = '''sto-3g''')
```

The molecular geometry is specified in Cartesian coordinates, with Angstrom as the default unit. The unit can be set using the `unit` attribute to either "Angstrom" or "Bohr".

```
mol.unit = 'B'
```

The attribute `mol.unit` is case-insensitive. Any string starting with "B" or "AU" is interpreted as Bohr; all other values are treated as Angstrom.

The input parser supports both Cartesian and Z-matrix formats, as well as explicit nuclear charges. Atoms can be labeled with numbers or special characters to distinguish them, allowing different basis sets, masses, or nuclear models to be assigned. Geometries can also be defined directly in the internal format of `Mole.atom` using Python lists, tuples, or NumPy arrays, or read from an .xyz file. For details, see: `https://pyscf.org/user/gto.html`.
After building the `Mole` object, the geometry can be accessed with `Mole.atom_coords()`, which returns an (N, 3) array of atomic positions. Note that the default unit is Bohr.

```
print(mol.atom_coords(unit='Bohr'))
[[0.  0.  0.]
 [0.  1.  0.]
 [0.  0.  1.]]
```

Basis sets

Assigning a basis set name (string) to `Mole.basis` applies it to all atoms. If different basis sets are needed for different elements, `Mole.basis` should be specified as a Python dictionary.

```
mol.basis = {'O': 'sto-3g', 'H': '6-31g'}
```

Custom basis sets can be defined with `gto.basis.parse()`, which parses a string in NWChem format (`https://www.basissetexchange.org/`).

```
mol.basis = {'O': gto.basis.parse('''
C    S
     71.6168370              0.15432897
     13.0450960              0.53532814
      3.5305122              0.44463454
C    SP
      2.9412494             -0.09996723             0.15591627
      0.6834831              0.39951283             0.60768372
      0.2222899              0.70011547             0.39195739
''')}
```

As with geometry input, the basis sets dictionary can use either atomic symbols (case-insensitive) or atomic nuclear charges. Numbers or special characters may be prefixed or suffixed to atom labels. If a decorated symbol is not found in the dictionary, the parser removes the decoration and looks up the plain atomic symbol.

In the following example, the `6-31G` basis set will be assigned to the atom `H1`, but the `STO-3G` basis will be used for the atom `H2`:

```
mol.atom = '8 0 0 0; H1 0 1 0; H2 0 0 1'
mol.basis = {'O': 'sto-3g', 'H': 'sto-3g', 'H1': '6-31G'}
```

Basis sets format

Basis sets data should be supplied in NWChem format, either as a text file or a Python script. PySCF converts them in the internal format, which looks like:

```
[[angular, kappa, [[exp, c_1, c_2, ..],
                   [exp, c_1, c_2, ..],
                   ... ]],
[angular, kappa, [[exp, c_1, c_2, ..],
                   [exp, c_1, c_2, ..]
                   ... ]]]
```

where `angular` is the angular momentum, `kappa` are the Gaussian exponents and `c` the contraction coefficients. `kappa` can have values; $-l-1$ (corresponding to spinors with $j = l + 1/2$), l (corresponding to spinors with $j = l - 1/2$) or 0. When `kappa` is 0, both types of spinors are assumed in the basis.

GTO basis functions are stored in the following order:

- atoms
- angular momentum
- shells
- spherical harmonics

Basis functions are first grouped by the atoms to which they belong. Within each atom, they are organized by angular momentum. For a given angular momentum, shells are ordered from inner to outer, i.e., from large to small exponents. A shell may represent a true atomic shell (a linear combination of Gaussians) or a single primitive Gaussian with multiple angular components.

DFT calculations

DFT is a self-consistent field method, therefore a self-consistent iterative procedure is applied, which begins from some initial guess. Kohn-Sham DFT has been implemented through derived classes of the `pyscf.scf.hf.SCF` parent class. As such, the methods and capabilities introduced in self-consistent field (SCF) methods are also available to the `dft` module.

Let's run our first DFT calculation!

Perform a DFT calculation of the hydrogen fluoride (HF) molecule to calculate its energy.

```
from pyscf import gto, dft
mol_hf = gto.M(atom = 'H 0 0 0; F 0 0 1.1', basis = 'ccpvdz', symmetry = True)
mf_hf = dft.RKS(mol_hf)
mf_hf.xc = 'lda,vwn' # default
mf_hf.kernel()
```

`symmetry`: Point group symmetry information
The point group symmetry information is held in the `Mole` object. The symmetry module of PySCF can detect arbitrary point groups.

Check the detected point group of the system, using:

```
print(mol.topgroup)
```

If `Mole.symmetry` is set, `Mole.build()` checks that the geometry satisfies the specified symmetry. If it does not, initialization stops and an error is raised.

`.kernel()`: Function to call HF driver.

Re-run the calculation with different basis sets. For details, consult `https://www.basissetexchange.org/`.

Evaluate the effect of different exchange-correlation functionals.

```
from pyscf import gto
from pyscf import dft

mol = gto.M(
    atom = '''
    O  0.    0.        0.
    H  0.    -0.757    0.587
    H  0.    0.757     0.587 ''',
    basis = 'ccpvdz')

mf = dft.RKS(mol)
# B3LYP can be constructed
mf.xc = 'HF*0.2 + .08*LDA + .72*B88, .81*LYP + .19*VWN'
e1 = mf.kernel()
print('E = %.15g  ref = -76.3832244350081' % e1)

# A shorthand to input 'PBE,PBE', which is a compound functional. Note the
# shorthand input is different to the two examples 'PBE,' and ',PBE' below.
mf.xc = 'PBE'

# Exchange part only
mf.xc = 'PBE,'

# Correlation part only
mf.xc = ',PBE'
```

`.verbose`: Control of the print level globally. The print level can be 0 (quiet, no output) to 9 (very noisy).

Perform a DFT calculation of H_2O molecule using B3LYP functional.

Geometry optimization

Perform geometry optimization of N_2 molecule.

Install `geomeTRIC` and `PyBerny`.

```
pip install geometric
pip install pyberny
```

```
from pyscf import gto, dft
from pyscf.geomopt.geometric_solver import optimize
from pyscf.geomopt.berny_solver import optimize

mol = gto.M(atom='N 0 0 0; N 0 0 1.2', basis='ccpvdz')
mf = dft.RKS(mol)
mf.xc = ',PBE'

# geometric
from pyscf.geomopt.geometric_solver import optimize
mol_eq = optimize(mf, maxsteps=100)
print(mol_eq.tostring())

# pyberny
from pyscf.geomopt.berny_solver import optimize
mol_eq = optimize(mf, maxsteps=100)
print(mol_eq.tostring())
```

Perform a geometry optimization calculation for the H_2O you used above.

Submit jobs with slurm

Rather than running interactively, jobs can be submitted via slurm using a submission script. Example for MeluXina:

```
#!/bin/bash -l
#SBATCH --qos=default                       # SLURM qos
#SBATCH --nodes=1                           # number of nodes
#SBATCH --ntasks=1                          # number of tasks
#SBATCH --ntasks-per-node=1                 # number of tasks per node
#SBATCH --time=00:05:00                     # time (HH:MM:SS)
#SBATCH --partition=cpu                     # partition
#SBATCH -A p201028                          # project account
#SBATCH --cpus-per-task=1                   # CORES per task

python3 geom.py > out
```

Save the above commands in a plain text file. If you are not familiar with other text editors, you can use `vi`. You can open a new file, by typing `vi` and the name of the file:

```
vi job_script.sh
```

After hitting `enter` you will find yourself in `vi`. `vi` has three different working modes:

- Normal mode: If you start the program or if you hit `Esc` buttom. In this mode, you can give commands to `vi`, as specified later.

- Insert mode: By typing `i`, you will be in insert mode. If you type something now, it will be inserted in your file.
- Replace mode: By hitting `i` again in the insert mode, you will pass to replace mode. If you start typing, `vi` will replace the letters below your cursor with a new one, rather than adding them. By hitting `i` again, you switch back to regular insert mode.

In your freshly opened file switch to insert mode by pressing `i`. Then copy-paste the above commands. To save the file, you have to exit the insert mode by pressing `Esc`. In escape mode, type `:w` to save the file, or type `:w job_script.sh` to save it under a filename called `job_script.sh`. You can exit `vi` by typing `:q`. If there were changes since your last save, type `:q!`, where the `!` forces `vi` to quit, without applying the changes.

To perform the geometry optimization via slurm rather than interactively, the PySCF commands should be saved in a ***`python`*** *script file, as well. Try running the N_2 example from above using slurm, too.*