Generating AFQMC Input

Fionn Malone

QMCPACK Users Meeting

14th May 2019

LLNL-PRES-XXXXXX

CPSFM | Center for Predictive Simulation of Functional Materials

Lawrence Livermore National Laboratory

Introduction

- AFQMC simulations are relatively straightforward.
- Need h_{ij}, L^n_{ik} and $|\phi_T\rangle$ ($|\phi_0\rangle$).
- In principle can come from any code.
- In practice we use PYSCF.
- In `$HOME/apps/qmcpack/qmcpack/utils/afmqctools` you will find a collection of tools to generate AFQMC input data.
- In `afqmctools/bin` you will find helper script: `pyscf_to_afqmc.py`.

pyscf_to_afqmc.py: 01-Neon_atom

Step 1 Run pyscf SCF calculation.

```
from pyscf import gto, scf

mol = gto.Mole()
mol.basis = 'aug-cc-pvdz'
mol.atom = (('Ne', 0,0,0),)
mol.verbose = 4
mol.build()

mf = scf.RHF(mol)
mf.chkfile = 'scf.chk'
ehf = mf.kernel()
```

pyscf_to_afqmc.py

Step 2 Run `pyscf_to_afqmc.py`:

```
pyscf_to_afqmc.py -i scf.chk -o hamil.h5 -t 1e-5 -q afqmc.xml
```

(`$HOME/apps/qmcpack/qmcpack/utils/afqmcutils/bin/pyscf_to_afqmc.py`)

Step 3 Run QMCPACK:

```
mpirun -n 8 qmcpack afqmc.xml
```

(`$HOME/apps/qmcpack/build_complex/bin/qmcpack`)

Solids: 06-diamond_2x2x2_supercell

- Run PYSCF scf calculation as before, except we store fock matrix and orthogonalisation matrix $X[k]$.

```
mf = scf.KRHF(cell, kpts=kpts)
mf.chkfile = 'scf.chk'
mf.kernel()

from afqmctools.utils.linalg import get_ortho_ao
hcore = mf.get_hcore()
fock = (hcore + mf.get_veff())
X, nmo_per_kpt = get_ortho_ao(cell,kpts)
with h5py.File(mf.chkfile) as fh5:
  fh5['scf/hcore'] = hcore
  fh5['scf/fock'] = fock
  fh5['scf/orthoAORot'] = X
  fh5['scf/nmo_per_kpt'] = nmo_per_kpt
```

- Integral generation proceeds as before, pass `-k/--kpoint` to use kpoint symmetry.

pyscf_to_afqmc.py

```
> pyscf_to_afqmcp.py --help
usage: pyscf_to_afqmc.py [-h] [-i CHK_FILE] [-o HAMIL_FILE] [-q QMC_INPUT]
                         [-t THRESH] [-k] [-g] [-a] [-c CAS] [-v]

optional arguments:
  -h, --help            show this help message and exit
  -i CHK_FILE, --input CHK_FILE
                        Input pyscf .chk file.
  -o HAMIL_FILE, --output HAMIL_FILE
                        Output file name for QMCPACK hamiltonian.
  -q QMC_INPUT, --qmcpack-input QMC_INPUT
                        Generate skeleton QMCPACK input file.
  -t THRESH, --cholesky-threshold THRESH
                        Cholesky convergence threshold.
  -k, --kpoint          Generate explicit kpoint dependent integrals.
  -g, --gdf             Use Gaussian density fitting.
  -a, --ao              Transform to ortho AO basis. Default assumes we work
                        in MO basis
  -c CAS, --cas CAS     Specify a CAS in the form of N,M.
  -v, --verbose         Verbose output.
```

Tutorials

- Tutorials are split into 7 examples.

```
01-neon_atom
02-neon_frozen_core
03-carbon_triplet_uhf
04-trial_wavefunction
05-methane_converge_back_prop
06-diamond_2x2x2_supercell
07-diamond_2x2x2_kpoint_sym
```

afqmctools as a library: pyscf tools

- Most advanced functions depend on pyscf.
- See `04-trial_wavefunction`.
- Another example:

```
from afqmcutils.utils.pyscf_utils import load_from_pyscf_chk_mol
from afqmcutils.wfn.mol import write_wfn_mol
from afqmcutils.hamiltonian.mol import write_hamil_mol
chkfiles = ['scf.rhf.chk','scf.uhf.chk','scf.pbe.chk']

scf_data = load_from_pyscf_chk_mol('scf.rhf.chk')
ortho_ao = True
write_hamil_mol(scf_data, 'hamil.h5', 1e-5,
                verbose=True, ortho_ao=ortho_ao)
for chk in chkfiles:
        scf_data = load_from_pyscf_chk_mol(chk)
        mf_type = chk.split('.')[1]
        write_wfn_mol(scf_data, 'hamil.h5', 1e-5, verbose=True,
                      ortho_ao=ortho_ao, wfn_name=mf_type)
```

- WARNING: Use same orthogonalisation matrix.

afqmctools as a library: User defined Hamiltonian

- Assuming your favourite code can generate Cholesky (and hcore) you can use afqmctools:

```
nalpha = 5
nbeta = 5
nmo = 10
hcore = numpy.random.rand(nmo*nmo).reshape(nmo,nmo)
hcore = 0.5 * (hcore + hcore.T)
eri = numpy.random.rand(nmo**4).reshape(nmo*nmo,nmo*nmo)
eri = numpy.dot(eri,eri.T)
chol = numpy.linalg.cholesky(eri)
chol_sparse = scipy.linalg.csr_matrix(chol[:,:50])
enuc = -1094.2
write_qmcpack_cholesky(hcore, chol_sparse, (nalpha,nbeta), nmo,
                       enuc=enuc, filename='hamil.h5')
```

`afqmctools` as a library: User defined Wavefunction

- Should be straightforward to convert wavefunction from any format through python.

```
nalpha = 5
nbeta = 5
nelec = nalpha + nbeta
nmo = 10
wfn = numpy.random.rand(nmo*nelec).reshape((1,nmo,nelec))
uhf = True
write_nomsd_wfn('crazy.dat', wfn, nalpha, uhf)
```

`fcidump_to_afqmc.py`: Or use FCIDUMP format

- Also included in `afqmctools/bin`
- Assumes 8-fold symmetric (real) integrals.
- Not advised, but may be simplest route for certain codes.

Summary

- `pyscf_to_afqmc.py` is your friend.
- Code is mostly stable but some features are deprecated.
- Basic functionality will remain.
- Plain text wavefunctions will disappear.
- Open issues on github.
- Acknowledgements: ECP/CPSFM.