PySCF and its toolkit for excited states

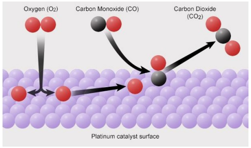
Qiming Sun

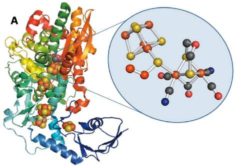
California Institute of Technology

June 9, 2018

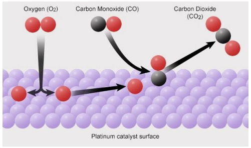
http://www.pyscf.org

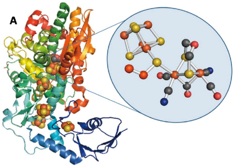
Progress in Science





Progress in Science





Density matrix embedding theory (DMET)

$$\begin{tikzcd}
\text{Mean-field framework} \arrow[rr] & & \text{Embedding environment} \arrow[dd] \\
& \text{Self-consistent iterations} & \\
\text{Embedding potential} \arrow[uu] & & \text{Impurity solver} \arrow[ll]
\end{tikzcd}$$

Electronic structure solvers

CAS

VASP

NWCHEM
HIGH-PERFORMANCE COMPUTATIONAL CHEMISTRY SOFTWARE

PSI4
OPEN-SOURCE QUANTUM CHEMISTRY

CP2K

DFTB

$\langle cc|cc \rangle$

QUANTUMESPRESSO

ORCA

SCM
Scientific Computing & Modelling

Ψ
ACES QC

abinit

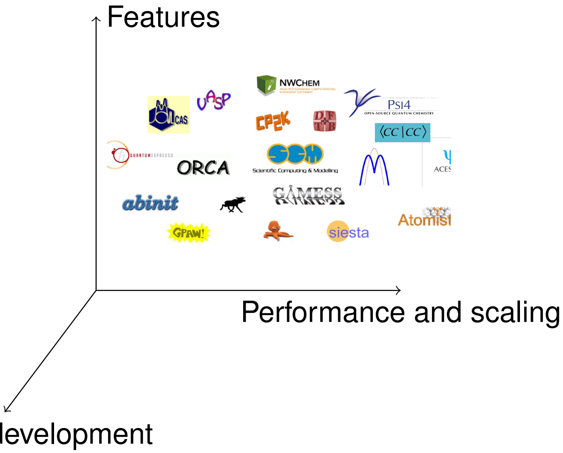
GAMESS

Atomistica

GPAW!

siesta

Software development strategy

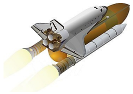


PySCF

Python simulations of chemistry framework

Complexity

Features


Performance

Installation

- `pip install pyscf`
- `conda install -c pyscf pyscf`
- `docker pull pyscf/pyscf-1.5.0`
- `https://www.pyscf.org/install.html`

PySCF releases

- v1.0 (Oct 2015)
  - SCF, MCSCF, MP2, CISD, CCSD, MRPT, DMRG-MCSCF
  - AO-Integral and MO-integral APIs
- v1.2 (Nov 2016)
  - CCSD, CCSD nuclear gradients, EOM-CCSD
- v1.3 (Apr 2017)
  - Periodic boundary condition
- v1.4 (Oct 2017)
  - post-HF method under PBC
  - Performance of PBC module
- v1.5 (Jun 2018)
  - Analytical nuclear gradients for ground and excited states

PySCF future release plans

- v1.6 (Dec 2018)
  - Empirical and semi-empirical models
- v1.7 (Jun 2019)
  - First order wavefunction
  - Electric and magnetic properties
- v1.8 (Dec 2019)
  - Relativistic methods
- v1.9 (Jun 2020)
  - Local correlation

Features for excited states as of PySCF-1.5

- Ground state energy

```
from pyscf import gto
mol = gto.M(atom="N 0 0 0; N 0 0 1.2", basis="ccpvdz")
ci = mol.apply("CISD").run()
ks = mol.apply("RKS").run()
```

- Excited states

```
ci.nstates = 5
ci.run()

from pyscf import tdscf
td = tdscf.TDRKS(ks)
td.nstates = 5
td.run()
```

Nuclear gradients for excited states

- Ground state nuclear gradients

```
force = ci.nuc_grad_method().kernel()
force = ks.nuc_grad_method().kernel()
```

- Nuclear gradients of excited states

```
ci.nstates = 5
force = ci.nuc_grad_method().kernel(state=1)

td = ks.apply("TDRKS")
td.nstates = 5
force = td.nuc_grad_method().kernel(state=1)
```

Scanner

- Energy and gradients for multiple geometries

```
from pyscf import gto
mol = gto.M(atom="N; N 1 1.2", basis="ccpvdz")
ci_scan = mol.apply("CASCI", 8, 10).as_scanner()
e, force = ci_scan("N; N 1 1.3")
e, force = ci_scan("N; N 1 1.4")

grad_scan = mc.nuc_grad_method().as_scanner()
e, force = grad_scan("N; N 1 1.3")
e, force = grad_scan("N; N 1 1.4")

# The 4th excited state
grad_scan = mc.nuc_grad_method().as_scanner(state=4)
e, force = grad_scan("N; N 1 1.3")
e, force = grad_scan("N; N 1 1.4")
```

Transition properties

- Transition density matrix

```
from pyscf import gto
mol = gto.M(atom="N; N 1 1.2", basis="ccpvdz")
ci = mol.apply("CISD")
ci.nstates = 4
ci.run()
t_dm1 = myci.trans_rdm1(ci.ci[3], ci.ci[0])
```

- Transition dipole

```
td = mol.apply("RKS").apply("TDA")
td.nstates = 5
td.run()
t_dip = td.transition_dipole()
```

PySCF designs

- Numpy/scipy-style library
- Lightweight (Python/C: 90/10)

	Python	C
Overall	197 k	46.5 k
DFT	8.3 k	7.8 k
MCSCF	10.4 k	0
PBC	37.4 k	2.4 k
CCSD	21.2 k	2.0 k

- Parallelism
  - Threading only
  - MPI, spark (as plugin) for distributed parallelism

Accessing Hamiltonian

- One-electron and two-electron integrals

```
mol = gto.M(atom="N; N 1 1.2", basis="ccpvdz")
hcore = mol.intor("int1e_kin") + mol.intor("int1e_nuc")
eri = mol.intor("int2e")
with mol.with_common_origin([0,0,0.6]):
    dip = mol.intor("int1e_r")
```

- Integrals with periodic boundary condition

```
from pyscf.pbc import gto, df
cell = gto.M(...)
hcore_kpt = (cell.pbc_intor("int1e_kin", kpt=kpt)
             + df.FFTDF(cell).get_pp(kpt))
eri_kpt = df.FFTDF(cell).get_eri([kpt1,kpt2,kpt3,kpt4])
```

Accessing Hamiltonian

- Integral transformation

```
from pyscf import ao2mo
occ_orb = ks.mo_coeff[:,:7]
vir_orb = ks.mo_coeff[:,7:]
ovov = ao2mo.kernel(mol, [occ_orb,vir_orb,
                          occ_orb,vir_orb])
```

- Customized Hamiltonian

```
mol = gto.Mole()
mol.nelectron = 6
mf = scf.RHF(mol)
mf.get_hcore = lambda: hcore
mf.get_ovlp = lambda: overlap
mf._eri = eri
mf.kernel()
cc.CCSD(mf).kernel()
```

Macro for asynchronous tasks

- Overlaping IO and computation

```
with h5py.File('intermediates.h5') as f:
  d = f.create_dataset("vvvv", (nv,nv,nv,nv), "f8")
  with lib.call_in_background(d.__setitem__) as save:
    for p0, p1 in lib.prange(0, nv, block):
      v = einsum("ijab,ijcd->acbd", t2[:,:,p0:p1], t2)
      save(slice(p0,p1), v)
```

- Overlaping MPI communication and computation

```
with h5py.File('intermediates.h5') as f:
  with lib.call_in_background(mpi.bcast) as bcast:
    for p0, p1 in lib.prange(0, nvir, block):
      v = f["vvvv"][p0:p1]
      bcast(v)
```

Seamless MPI mode

Serial version

```
from pyscf.pbc import, scf
from pyscf.pbc import df
cell = ...
mf = scf.KRHF(cell)
mf.with_df = df.AFTDF(cell)
mf.kpts = cell.get_kpts([2]*3)
mf.kernel()
```

```
export OMP_NUM_THREADS=16
time python diamond.py
.. 1340% cpu 4:48.49 total
```

MPI version

```
from pyscf.pbc import, scf
from mpi4pyscf.pbc import df
cell = ...
mf = scf.KRHF(cell)
mf.with_df = df.AFTDF(cell)
mf.kpts = cell.get_kpts([2]*3)
mf.kernel()
```

```
export OMP_NUM_THREADS=4
time mpirun -n 4 python diamond.py
..  1495% cpu 4:10.46 total
```

```
export OMP_NUM_THREADS=2
time mpirun -n 8 python diamond.py
..  1501% cpu 4:08.19 total
```

Technical notes

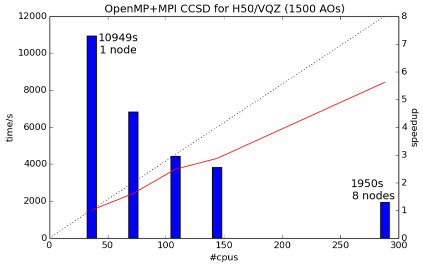
- Minimal library dependence
- API driven
- Test driven
- Global configs
- Server/client mode for parallel code
- `ctypes` for Fortran, C/C++ interfaces

Technical notes

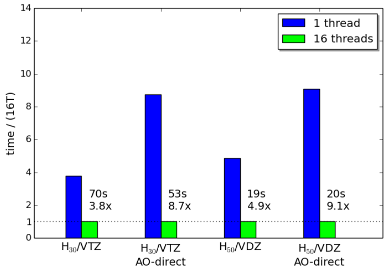
- Minimal library dependence
- API driven
- Test driven
- Global configs
- Server/client mode for parallel code
- `ctypes` for Fortran, C/C++ interfaces

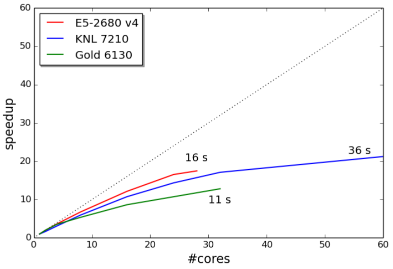
Designed and implemented as a toolkit for fast development

Performance



- OpenMP+MPI CCSD
- OpenMP CCSD
- OpenMP FCI(16,16)





Summary

PySCF 1.5 release

- Functions are available for energy, nuclear gradients, and properties of excited states.

Design of PySCF

- PySCF is desinged as a toolkit for fast development

Question

- How to call PySCF in the Fortran/C/C++ program?

Acknowledgements

- Prof. Garnet K.-L. Chan
- Prof. Timothy C. BerkelBach
- Prof. Sandeep Sharma
- Dr. James D. McClain
- Yanli Hu

Thank you for your attention!