Recent developments in the PySCF program package

Qiming Sun,[1] Xing Zhang,[2] Samragni Banerjee,[3] Peng Bao,[4] Marc Barbry,[5] Nick S. Blunt,[6] Nikolay A. Bogdanov,[7] George H. Booth,[8] Jia Chen,[9, 10] Zhi-Hao Cui,[2] Janus Juul Eriksen,[11] Yang Gao,[12] Sheng Guo,[13] Jan Hermann,[14, 15] Matthew R. Hermes,[16] Kevin Koh,[17] Peter Koval,[18] Susi Lehtola,[19] Zhendong Li,[20] Junzi Liu,[21] Narbe Mardirossian,[22] James D. McClain,[23] Mario Motta,[24] Bastien Mussard,[25] Hung Q. Pham,[16] Artem Pulkin,[26] Wirawan Purwanto,[27] Paul J. Robinson,[28] Enrico Ronca,[29] Elvira Sayfutyarova,[30] Maximilian Scheurer,[31] Henry F. Schurkus,[2] James E. T. Smith,[25] Chong Sun,[2] Shi-Ning Sun,[12] Shiv Upadhyay,[32] Lucas K. Wagner,[33] Xiao Wang,[34] Alec White,[2] James Daniel Whitfield,[35] Mark J. Williamson,[36] Sebastian Wouters,[37] Jun Yang,[38] Jason M. Yu,[39] Tianyu Zhu,[2] Timothy C. Berkelbach,[28, 34] Sandeep Sharma,[25] Alexander Yu. Sokolov,[3] and Garnet Kin-Lic Chan[2, a)]

1)*AxiomQuant Investment Management LLC, Shanghai, 200120, China*
2)*Division of Chemistry and Chemical Engineering, California Institute of Technology, Pasadena, CA 91125, USA*
3)*Department of Chemistry and Biochemistry, The Ohio State University, Columbus, OH 43210, USA*
4)*Beijing National Laboratory for Molecular Sciences, State Key Laboratory for Structural Chemistry of Unstable and Stable Species, Institute of Chemistry, Chinese Academy of Sciences, Beijing 100190, China*
5)*Simbeyond B.V., P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands*
6)*Department of Chemistry, Lensfield Road, Cambridge, CB2 1EW, United Kingdom*
7)*Max Planck Institute for Solid State Research, Heisenbergstraße 1, 70569 Stuttgart, Germany*
8)*Department of Physics, King's College London, Strand, London WC2R 2LS, United Kingdom*
9)*Department of Physics, University of Florida, Gainesville, FL 32611, USA*
10)*Quantum Theory Project, University of Florida, Gainesville, FL 32611, USA*
11)*School of Chemistry, University of Bristol, Cantock's Close, Bristol BS8 1TS, United Kingdom*
12)*Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125, USA*
13)*Google Inc., Mountain View, CA 94043, USA*
14)*FU Berlin, Department of Mathematics and Computer Science, Arnimallee 6, 14195 Berlin, Germany*
15)*TU Berlin, Machine Learning Group, Marchstr. 23, 10587 Berlin, Germany*
16)*Department of Chemistry, Chemical Theory Center, and Supercomputing Institute, University of Minnesota, 207 Pleasant Street SE, Minneapolis, MN 55455, USA*
17)*Department of Chemistry and Biochemistry, The University of Notre Dame du Lac, 251 Nieuwland Science Hall, Notre Dame, IN 46556, USA*
18)*Simune Atomistics S.L., Avenida Tolosa 76, Donostia-San Sebastian, Spain*
19)*Department of Chemistry, University of Helsinki, P.O. Box 55 (A. I. Virtasen aukio 1), FI-00014 Helsinki, Finland.*
20)*Key Laboratory of Theoretical and Computational Photochemistry, Ministry of Education, College of Chemistry, Beijing Normal University, Beijing 100875, China*
21)*Department of Chemistry, The Johns Hopkins University, Baltimore, MD 21218, USA*
22)*AMGEN Research, One Amgen Center Drive, Thousand Oaks, CA 91320, USA*
23)*DRW Holdings LLC, Chicago, IL 60661, USA*
24)*IBM Almaden Research Center, San Jose, CA 95120, USA*
25)*Department of Chemistry, University of Colorado, Boulder, CO 80302, USA*
26)*QuTech and Kavli Institute of Nanoscience, Delft University of Technology, The Netherlands*
27)*Information Technology Services, Old Dominion University, Norfolk, VA 23529, USA*
28)*Department of Chemistry, Columbia University, New York, NY 10027, USA*
29)*Istituto per i Processi Chimico Fisici del CNR (IPCF-CNR), Via G. Moruzzi, 1, 56124, Pisa, Italy*
30)*Department of Chemistry, Yale University, 225 Prospect Street, New Haven, CT 06520, USA*
31)*Interdisciplinary Center for Scientific Computing, Ruprecht-Karls University of Heidelberg, 205 Im Neuenheimer Feld, 69120 Heidelberg, Germany*
32)*Department of Chemistry, University of Pittsburgh, Pittsburgh, PA 15260*
33)*Department of Physics and Institute for Condensed Matter Theory, University of Illinois at Urbana-Champaign, IL 61801, USA*
34)*Center for Computational Quantum Physics, Flatiron Institute, New York, NY 10010, USA*
35)*Department of Physics and Astronomy, Dartmouth College, Hanover, NH 03755, USA*

[36]*Department of Chemistry, University of Cambridge, Lensfield Road, Cambridge CB2 1EW, United Kingdom*
[37]*Bricsys NV, Bellevue 5/201, 9050 Gent, Belgium*
[38]*Department of Chemistry, The University of Hong Kong, Pokfulam Road, Hong Kong SAR, China*
[39]*Department of Chemistry, University of California, Irvine, 1102 Natural Sciences II, Irvine, CA 92697-2025, USA*

PySCF is a Python-based general-purpose electronic structure platform that both supports first-principles simulations of molecules and solids, as well as accelerates the development of new methodology and complex computational workflows. The present paper explains the design and philosophy behind PySCF that enables it to meet these twin objectives. With several case studies, we show how users can easily implement their own methods using PySCF as a development environment. We then summarize the capabilities of PySCF for molecular and solid-state simulations. Finally, we describe the growing ecosystem of projects that use PySCF across the domains of quantum chemistry, materials science, machine learning and quantum information science.

I. INTRODUCTION

This article describes the current status of the Python Simulations of Chemistry Framework, also known as PySCF, as of version 1.7.1. The PySCF project was originally started in 2014 by Sun, then in the group of Chan, in the context of developing a tool to enable ab initio quantum embedding calculations. However, it rapidly outgrew its rather specialized roots to become a general purpose development platform for quantum simulations and electronic structure theory. The early history of PySCF is recounted in Ref. 1. Now, PySCF is a production ready tool that implements many of the most commonly used methods in molecular quantum chemistry and solid-state electronic structure. Since its inception, PySCF has been a free and open-source package hosted on Github,[2] and is now also available through pip,[3] conda,[4] and a number of other distribution platforms. It has a userbase numbering in the hundreds, and over 60 code contributors. Beyond chemistry and materials science, it has also found use in the areas of data science,[5,6] machine learning,[7–13] and quantum computing,[14–16] in both academia as well as in industry. To mark its transition from a code developed by a single group to a broader community effort, the leadership of PySCF was expanded in 2019 to a board of directors.[17]

While the fields of quantum chemistry and solid-state electronic structure are rich with excellent software,[18–25] the development of PySCF is guided by some unique principles. In order of priority:

1. PySCF should be more than a computational tool; it should be a development platform. We aim for users to be empowered to modify the code, implement their own methods without the assistance of the original developers, and incorporate parts of the code in a modular fashion into their own projects;
2. Unlike many packages which focus on either molecular chemistry or materials science applications, PySCF should support both equally, to allow calculations on molecules and materials to be carried out in the same numerical framework and with the same theoretical approximations;
3. PySCF should enable users outside of the chemical sciences (such as workers in machine learning and quantum information theory) to carry out quantum chemistry simulations.

In the rest of this article, we elaborate on these guiding principles of PySCF, describing how they have impacted the program design and implementation and how they can be used to implement new functionality in new projects. We provide a brief summary of the implemented methods and conclude with an overview of the PySCF ecosystem in different areas of science.

II. THE DESIGN PHILOSOPHY BEHIND PYSCF

All quantum simulation workflows naturally require some level of programming and customization. This may arise in simple tasks, such as scanning a potential energy surface, tabulating results, or automating input generation, or in more advanced use cases that include more substantial programming, such as with complex data processing, incorporating logic into the computational workflow, or when embedding customized algorithms into the computation. In either case, the ability to program with and extend one's simulation software greatly empowers the user. PySCF is designed to serve as a basic program library that can facilitate custom computational tasks and workflows, as well as form the starting point for the development of new algorithms.

To enable this, PySCF is constructed as a library of modular components with a loosely coupled structure. The modules provide easily reusable functions, with (where possible) simple implementations, and hooks are provided within the code to enable extensibility. Optimized and competitive performance is, as much as possible, separated out into a small number of lower level components which do not need to be touched by the user. We elaborate on these design choices below:

- *Reusable functions for individual suboperations*.

---

[a]Electronic mail: gkc1000@gmail.com

It is becoming common practice to provide a Python scripting interface for input and simulation control. However, PySCF goes beyond this by providing a rich set of Python APIs not only for the simulation models, but also for many of the individual sub-operations that compose the algorithms. For example, after input parsing, a mean-field Hartree-Fock (HF) or density functional theory (DFT) algorithm comprises many steps, including integral generation, guess initialization, assembling components of the Fock matrix and diagonalizing, and accelerating iterations to self-consistent convergence. All of these suboperations are exposed as PySCF APIs, enabling one to rebuild or modify the self-consistent algorithm at will. Similarly, APIs are exposed for other essential components of electronic structure algorithms, such as integral transformations, density fitting, Hamiltonian manipulation, various many-electron and Green's functions solvers, computation of derivatives, relativistic corrections, and so forth, in essence across all the functionality of PySCF. The package provides a large number of examples to demonstrate how these APIs can be used in customized calculations or methodology development.

With at most some simple initialization statements, the PySCF APIs can be executed at any place and in any order within a code without side-effects. This means that when implementing or extending the code, the user does not need to retain information on the program state, and can focus on the physical theory of interest. For instance, using the above example, one can call the function to build a Fock matrix from a given density matrix anywhere in the code, regardless of whether the density matrix in question is related to a larger simulation. From a programming design perspective, this is because within PySCF no implicit global variables are used and functions are implemented free of side effects (or with minimal side effects) in a largely functional programming style. The PySCF function APIs generally follow the NUMPY/SCIPY API style. In this convention, the input arguments are simple Python built-in datatypes or NUMPY arrays, avoiding the need to understand complex objects and structures.

- *Simple implementations*.

  Python is amongst the simplest of the widely-used programming languages and is the main implementation language in PySCF. Apart from a few performance critical functions, over 90% of PySCF is written in Python, with dependencies on only a small number of common external Python libraries (NUMPY, SCIPY, H5PY).

  Implementation language does not hide organizational complexity, however, and structural simplicity in PySCF is achieved via additional design choices. In particular, PySCF uses a mixed object oriented/functional paradigm: complex simulation data (e.g. data on the molecular geometry or cell parameters) and simulation models (e.g. whether a mean-field calculation is a HF or DFT one) are organized in an object oriented style, while individual function implementations follow a functional programming paradigm. Deep object inheritance is rarely used. Unlike packages where external input configuration files are used to control a simulation, the simulation parameters are simply held in the member variables of the simulation model object.

  Where possible, PySCF provides multiple implementations of the same algorithm with the same API: one is designed to be easy to read and simple to modify, and another is for optimized performance. For example, the full configuration interaction module contains both a slower but simpler implementation as well as heavily optimized implementations, specialized for specific Hamiltonian symmetries and spin types. The optimized algorithms have components that are written in C. This dual level of implementation mimics the Python convention of having modules in both pure Python and C with the same API (such as the PROFILE and CPROFILE modules of the Python standard library). It also reflects the PySCF development cycle, where often a simple reference Python implementation is first produced before being further optimized.

- *Easily modified runtime functionality*.

  In customized simulations, it is often necessary to modify the underlying functionality of a package. This can be complicated in a compiled program due to the need to consider detailed types and compilation dependencies across modules. In contrast, many parts of PySCF are easy to modify both due to the design of PySCF as well as the dynamic runtime resolution of methods and "duck typing" of Python. Generally speaking, one can modify functionality in one part of the code without needing to worry about breaking other parts of the package. For example, one can modify the HF module with a custom Hamiltonian without considering whether it will work in a DFT calculation; the program will continue to run so long as the computational task involves HF and post-HF methods. Further, Python "monkey patching" (replacing functionality at runtime) means that core PySCF routines can be overwritten without even touching the code base of the library.

- *Competitive performance*.

  In many simulations, performance is still the critical consideration. This is typically the reason for implementing code in compiled languages such as FORTRAN or C/C++. In PySCF, the performance gap between Python and compiled languages is partly removed by a heavy reliance on NUMPY and SCIPY, which provide Python APIs to optimized algorithms written in compiled languages. Additional optimization is achieved in PySCF with custom C implementations where necessary. Performance critical spots, which occur primarily in the integral and tensor operations, are implemented in C and heavily optimized. The use of additional C

The Journal of Chemical Physics ACCEPTED MANUSCRIPT
This is the author's peer reviewed, accepted manuscript. However, the online version of record will be different from this version once it has been copyedited and typeset.
PLEASE CITE THIS ARTICLE AS DOI:10.1063/5.0006074
AIP Publishing

libraries also allows us to achieve thread-level parallelism via OpenMP, bypassing Python's intrinsic multithreading limitations. Since a simulation can often spend over 99% of its runtime in the C libraries, the overhead due to the remaining Python code is negligible. The combination of Python with C libraries ensures PySCF achieves leading performance in many simulations.

III. A COMMON FRAMEWORK FOR MOLECULES AND CRYSTALLINE MATERIALS

Electronic structure packages typically focus on either molecular or materials simulations, and are thus built around numerical approximations adapted to either case. A central goal of PySCF is to enable molecules and materials to be simulated with common numerical approximations and theoretical models. Originally, PySCF started as a Gaussian atomic orbital (AO) molecular code, and was subsequently extended to enable simulations in a crystalline Gaussian basis. Much of the seemingly new functionality required in a crystalline materials simulation is in fact analogous to functionality in a molecular implementation, such as

1. Using a Bloch basis. In PySCF we use a crystalline Gaussian AO basis, which is analogous to a symmetry adapted molecular AO basis;
2. Exploiting translational symmetry by enforcing momentum conservation. This is analogous to handling molecular point group symmetries;
3. Handling complex numbers, given that matrix elements between Bloch functions are generally complex. This is analogous to the requirements of a molecular calculation with complex orbitals.

Other modifications are unique to the crystalline material setting, including:

1. Techniques to handle divergences associated with the long-ranged nature of the Coulomb interaction, since the classical electron-electron, electron-nuclear, and nuclear-nuclear interactions are separately divergent. In PySCF this is handled via the density fitting integral routines (see below) and by evaluating certain contributions using Ewald summation techniques;
2. Numerical techniques special to periodic functions, such as the fast Fourier transform (FFT), as well as approximations tailored to plane-wave implementations, such as certain pseudopotentials. PySCF supports mixed crystalline Gaussian and plane-wave expressions, using both analytic integrals as well as FFT on grids;
3. Techniques to accelerate convergence to the thermodynamic limit. In PySCF, such corrections are implemented at the mean-field level by modifying the treatment of the exchange energy, which is the leading finite-size correction.
4. Additional crystal lattice symmetries. Currently PySCF contains only experimental support for additional lattice symmetries.

In PySCF, we identify the three-index density fitted integrals as the central computational intermediate that allows us to largely unify molecular and crystalline implementations. This is because:

1. three-center "density-fitted" Gaussian integrals are key to fast implementations;
2. The use of the FFT to evaluate the potential of a pair density of AO functions, which is needed in fast DFT implementations with pseudopotentials,[26] is formally equivalent to density fitting with plane-waves;
3. The density fitted integrals can be adjusted to remove the Coulomb divergences in materials;[27]
4. three-index Coulomb intermediates are sufficiently compact that they can be computed even in the crystalline setting.

PySCF provides a unified density fitting API for both molecules and crystalline materials. In molecules, the auxiliary basis is assumed to be Gaussian AOs, while in the periodic setting, different types of auxiliary bases are provided, including plane-wave functions (in the FFTDF module), crystalline Gaussian AOs (in the GDF module) and mixed plane-wave-Gaussian functions (in the MDF module).[28] Different auxiliary bases are provided in periodic calculations as they are suited to different AO basis sets: FFTDF is efficient for smooth AO functions when used with pseudopotentials; GDF is more efficient for compact AO functions; and MDF allows a high accuracy treatment of the Coulomb problem regardless of the compactness of the underlying atomic orbital basis.

Using the above ideas, the general program structure, implementation, and simulation workflow for molecular and materials calculations become very similar. Figure 1 shows an example of the computational workflow adopted in PySCF for performing molecular and periodic post-HF calculations. The same driver functions can be used to carry out generic operations such as solving the HF equations or coupled cluster amplitude equations. However, the implementations of methods for molecular and crystalline systems bifurcate when evaluating *k*-point dependent quantities, such as the three-center density-fitted integrals, Hamiltonians, and wavefunctions. Nevertheless, if only a single *k*-point is considered (and especially at the Γ point where all integrals are real), most molecular modules can be used to perform calculations in crystals without modification (see Sec. V).

IV. DEVELOPING WITH PYSCF: CASE STUDIES

In this section we walk through some case studies that illustrate how the functionality of PySCF can be modified and extended. We focus on cases which might be encountered by the average user who does not want to modify the source code,

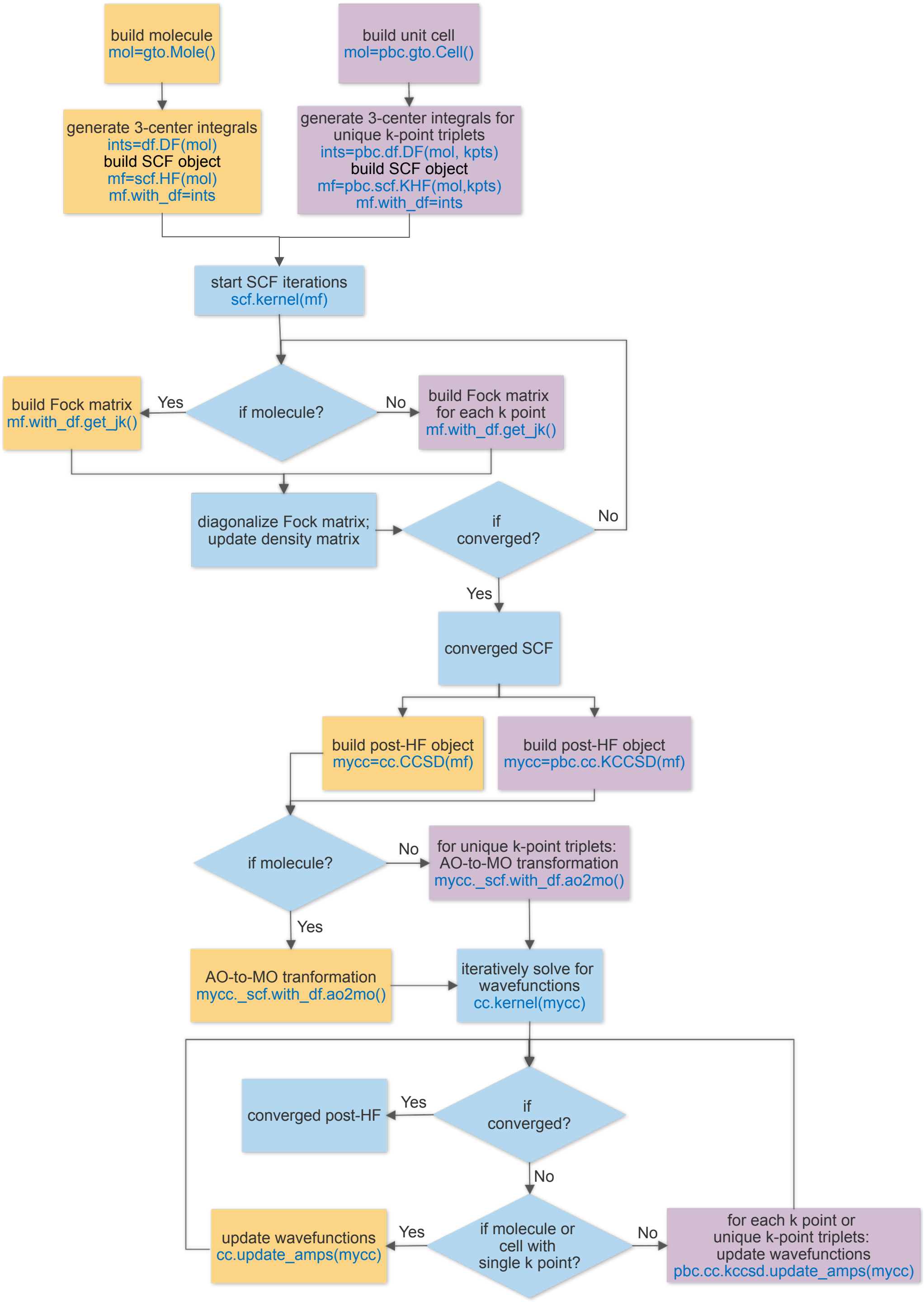

build molecule
mol=gto.Mole()
build unit cell
mol=pbc.gto.Cell()
generate 3-center integrals
ints=df.DF(mol)
build SCF object
mf=scf.HF(mol)
mf.with_df=ints
generate 3-center integrals for unique k-point triplets
ints=pbc.df.DF(mol, kpts)
build SCF object
mf=pbc.scf.KHF(mol,kpts)
mf.with_df=ints
start SCF iterations
scf.kernel(mf)
build Fock matrix
mf.with_df.get_jk()
Yes
if molecule?
No
build Fock matrix for each k point
mf.with_df.get_jk()
diagonalize Fock matrix; update density matrix
if converged?
No
Yes
converged SCF
build post-HF object
mycc=cc.CCSD(mf)
build post-HF object
mycc=pbc.cc.KCCSD(mf)
if molecule?
No
for unique k-point triplets: AO-to-MO transformation
mycc._scf.with_df.ao2mo()
Yes
AO-to-MO tranformation
mycc._scf.with_df.ao2mo()
iteratively solve for wavefunctions
cc.kernel(mycc)
converged post-HF
Yes
if converged?
No
update wavefunctions
cc.update_amps(mycc)
Yes
if molecule or cell with single k point?
No
for each k point or unique k-point triplets: update wavefunctions
pbc.cc.kccsd.update_amps(mycc)

FIG. 1: Illustration of the program workflow for molecular and periodic calculations. The orange and purple boxes indicate functions that are *k*-point independent and *k*-point dependent, respectively; the blue boxes indicate generic driver functions that can be used in both molecular and periodic calculations.

but wishes to assemble different existing PySCF APIs to implement new functionality.

A. Case study: modifying the Hamiltonian

In PySCF, simulation models (i.e. different wavefunction approximations) are always implemented such that they can be used independently of any specific Hamiltonian, with up to two-body interactions. Consequently, the Hamiltonian under study can be easily customized by the user, which is useful for studying model problems or, for example, when trying to interface to different numerical basis approximations. Figure 2 shows several different ways to define one-electron and two-electron interactions in the Hamiltonian followed by subsequent ground and excited state calculations with the custom Hamiltonian. Note that if a method is not compatible with or well defined using the customized interactions, for instance, in the case of solvation corrections, PySCF will raise a Python runtime error in the place where the requisite operations are ill-defined.

B. Case study: optimizing orbitals of arbitrary methods

The PySCF MCSCF module provides a general purpose quasi-second order orbital optimization algorithm within orbital subspaces (e.g. active spaces) as well as over the complete orbital space. In particular, it is not limited to the built-in CASCI, CASSCF and multi-reference correlation solvers, but allows orbital optimization of any method that provides energies and one- and two-particle density matrices. For this reason, PySCF is often used to carry out active space orbital optimization for DMRG (density matrix renormalization group), selected configuration interaction, and full configuration interaction quantum Monte Carlo wavefunctions, via its native interfaces to Block[29] (DMRG), CheMPS2[30] (DMRG), Dice[31–33] (selected CI), Arrow[31,33,34] (selected CI), and NECI[35] (FCIQMC).

In addition, it is easy for the user to use the MCSCF module to optimize orbitals in electronic structure methods for which the orbital optimization API is not natively implemented. For example, although orbital-optimized MP2[36] is not explicitly provided in PySCF, a simple version of it can easily be performed using a short script, shown in Figure 3. Without any modifications, the orbital optimization will use a quasi-second order algorithm. We see that the user only needs to write a simple wrapper to provide two functions, namely, `make_rdm12`, which computes the one- and two-particle density matrices, and `kernel`, which computes the total energy.

C. Case study: implementing an embedding model

As a more advanced example of customization using PySCF, we now illustrate how a simple script with standard APIs enables PySCF to carry out geometry optimization for a wavefunction in Hartree-Fock (WFT-in-HF) embedding model, shown in Figure 4 with a CISD solver. Given the Hamiltonian of a system, expressed in terms of the Hamiltonians of a fragment and its environment

$$\begin{aligned} H_{\text{sys}} &= H_{\text{frag}} + H_{\text{env}} + V_{\text{ee,frag-env}}, \\ H_{\text{frag}} &= h_{\text{core,frag}} + V_{\text{ee,frag}}, \\ H_{\text{env}} &= h_{\text{core,env}} + V_{\text{ee,env}}, \end{aligned}$$

we define an embedding Hamiltonian for the fragment in the presence of the atoms in the environment as

$$\begin{aligned} H_{\text{emb}} &= h_{\text{eff,frag}} + V_{\text{ee,frag}}, \\ h_{\text{eff,frag}} &= h_{\text{core,frag}} + (h_{\text{core,env}} + V_{\text{eff}}[\rho_{\text{env}}]), \\ V_{\text{eff}}[\rho_{\text{env}}] &= \int V_{\text{ee,env}}\rho_{\text{env}}(\mathbf{r})d\mathbf{r} + \int V_{\text{ee,frag-env}}\rho_{\text{env}}(\mathbf{r})d\mathbf{r} \end{aligned}$$

Geometry optimization can then be carried out with the approximate nuclear gradients of the embedding problem

$$\begin{aligned} \text{Gradients} &= \langle\Psi_{\text{CI}}|\frac{\partial H_{sys}}{\partial X}|\Psi_{\text{CI}}\rangle \\ &\approx \langle\Psi_{\text{CI}}|\frac{\partial H_{\text{frag}}}{\partial X}|\Psi_{\text{CI}}\rangle + \langle\Psi_{\text{HF}}|\frac{\partial(H_{\text{env}} + V_{\text{ee,frag-env}})}{\partial X}|\Psi_{\text{HF}}\rangle \\ &= \langle\Psi_{\text{CI}}|\frac{\partial H_{\text{frag}}}{\partial X}|\Psi_{\text{CI}}\rangle - \langle\Psi_{\text{HF}}|\frac{\partial H_{\text{frag}}}{\partial X}|\Psi_{\text{HF}}\rangle + \langle\Psi_{\text{HF}}|\frac{\partial H_{\text{sys}}}{\partial X}|\Psi_{\text{HF}}\rangle \\ &\approx \langle\Psi_{\text{frag,CI}}|\frac{\partial H_{\text{frag}}}{\partial X}|\Psi_{\text{frag,CI}}\rangle - \langle\Psi_{\text{frag,HF}}|\frac{\partial H_{\text{frag}}}{\partial X}|\Psi_{\text{frag,HF}}\rangle + \langle\Psi_{\text{HF}}|\frac{\partial H_{\text{sys}}}{\partial X}|\Psi_{\text{HF}}\rangle, \end{aligned}$$

where the fragment wavefunction $\Psi_{\text{frag,HF}}$ and $\Psi_{\text{frag,CI}}$ are obtained from the embedding Hamiltonian H_{emb}. The code snippet in Figure 4 demonstrates the kind of rapid prototyping that can be carried out using PySCF APIs. In particular, this demonstration combines the APIs for ab initio energy evaluation, analytical nuclear gradient computation, computing the HF potential for an arbitrary density matrix, Hamiltonian customization, and customizing the nuclear gradient solver in a geometry optimization.

V. SUMMARY OF EXISTING METHODS AND RECENT ADDITIONS

In this section we briefly summarize major current capabilities of the PySCF package. These capabilities are listed in

```
import numpy
import pyscf
mol = pyscf.M()
n = 10
mol.nelectron = n

# Define model Hamiltonian: tight binding on a ring
h1 = numpy.zeros((n, n))
for i in range(n-1):
    h1[i, i+1] = h1[i+1, i] = -1.
h1[n-1, 0] = h1[0, n-1] = -1.

# Build the 2-electron interaction tensor starting from a random 3-index tensor.
tensor = numpy.random.rand(2, n, n)
tensor = tensor + tensor.transpose(0, 2, 1)
eri = numpy.einsum('xpq,xrs->pqrs', tensor, tensor)

# SCF for the custom Hamiltonian
mf = mol.HF()
mf.get_hcore = lambda *args: h1
mf.get_ovlp = lambda *args: numpy.eye(n)

# Option 1: overwrite the attribute mf._eri for the 2-electron interactions
mf._eri = eri
mf.run()

# Option 2: introduce the 2-electron interaction through the Cholesky decomposed tensor.
dfmf = mf.density_fit()
dfmf.with_df._cderi = tensor
dfmf.run()

# Option 3: define a custom HF potential method
def get_veff(mol, dm):
    J = numpy.einsum('xpq,xrs,pq->rs', tensor, tensor, dm)
    K = numpy.einsum('xpq,xrs,qr->ps', tensor, tensor, dm)
    return J - K * .5
mf.get_veff = get_veff
mf.run()

# Call the second order SCF solver in case converging the DIIS-driven HF method
# without a proper initial guess is difficult.
mf = mf.newton().run()

# Run post-HF methods based on the custom SCF object
mf.MP2().run()
mf.CISD().run()
mf.CCSD().run()
mf.CASSCF(4, 4).run()
mf.CASCI(4, 4).run().NEVPT2().run()
mf.TDHF().run()
mf.CCSD().run().EOMIP().run()
mc = shci.SHCISCF(mf, 4, 4).run()
mc = dmrgscf.DMRGSCF(mf, 4, 4).run()
```

FIG. 2: Hamiltonian customization and post-HF methods for customized Hamiltonians.

Table I and details are presented in the following subsections.

A. Hartree-Fock and density functional theory methods

The starting point for many electronic structure simulations is a self-consistent field (SCF) calculation. PySCF implements Hartree-Fock (HF) and density functional theory (DFT) with a variety of Slater determinant references, including restricted closed-shell, restricted open-shell, unrestricted, and generalized (noncollinear spin) references,[37,38] for both molecular and crystalline (*k*-point) calculations. Through an interface to the Libxc[39] and XCFun[40] libraries, PySCF also supports a wide range of predefined exchange-correlation (XC) functionals, including the local density approximation (LDA), generalized gradient approximations (GGA), hybrids,

```python
import numpy
import pyscf
class MP2AsFCISolver(object):
    def kernel(self, h1, h2, norb, nelec, ci0=None, ecore=0, **kwargs):
        # Kernel takes the set of integrals from the current set of orbitals
        fakemol = pyscf.M(verbose=0)
        fakemol.nelectron = sum(nelec)
        fake_hf = fakemol.RHF()
        fake_hf._eri = h2
        fake_hf.get_hcore = lambda *args: h1
        fake_hf.get_ovlp = lambda *args: numpy.eye(norb)

        # Build an SCF object fake_hf without SCF iterations to perform MP2
        fake_hf.mo_coeff = numpy.eye(norb)
        fake_hf.mo_occ = numpy.zeros(norb)
        fake_hf.mo_occ[:fakemol.nelectron//2] = 2
        self.mp2 = fake_hf.MP2().run()
        return self.mp2.e_tot + ecore, self.mp2.t2

    def make_rdm12(self, t2, norb, nelec):
        dm1 = self.mp2.make_rdm1(t2)
        dm2 = self.mp2.make_rdm2(t2)
        return dm1, dm2

mol = pyscf.M(atom=’H 0 0 0; F 0 0 1.1’, basis=’ccpvdz’)
mf = mol.RHF().run()
# Put in the active space all orbitals of the system
mc = pyscf.mcscf.CASSCF(mf, mol.nao, mol.nelectron)
mc.fcisolver = MP2AsFCISolver()
# Internal rotation inside the active space needs to be enabled
mc.internal_rotation = True
mc.kernel()
```

FIG. 3: Using the general CASSCF solver to implement an orbital-optimized MP2 method.

meta-GGAs, nonlocal correlation functionals (VV10[41]) and range-separated hybrid (RSH) functionals. In addition to predefined XC functionals, the user can also create customized functionals in a DFT calculation, as shown in Figure 5.

Because PySCF uses a Gaussian AO representation, the SCF computation is usually dominated by Gaussian integral evaluation. Through the efficient Gaussian integral engine Libcint,[42] the molecular SCF module can be used with more than 10,000 basis functions on a symmetric multiprocessing (SMP) machine, without resorting to any integral approximations such as screening. Further speed-up can be achieved through Gaussian density fitting, and the pseudo-spectral approach (SGX) is implemented to speed up the evaluation of exchange in large systems.[43–45]

In crystalline systems, HF and DFT calculations can be carried out either at a single point in the Brillouin zone or with a *k*-point mesh. The cost of the crystalline SCF calculation depends on the nature of the crystalline Gaussian basis and the associated density fitting. PySCF supports Goedecker-Teter-Hutter (GTH) pseudopotentials[46] which can be used with the associated basis sets (developed by the CP2K group).[26,47] Pseudopotential DFT calculations are typically most efficiently done using plane-wave density fitting (FFTDF). Alternatively, all-electron calculations can be performed with standard basis sets, and the presence of sharp densities means that Gaussian density fitting performs better. Gaussian density fitting is also the algorithm of choice for calculations with HF exchange. Figure 6 shows an example of the silicon band structures computed using a GTH-LDA pseudopotential with FFTDF, and in an all-electron calculation using GDF.

B. Many-body methods

Starting from a SCF HF or DFT wavefunction, various many-body methods are available in PySCF, including Møller-Plesset second-order perturbation theory (MP2), multi-reference perturbation theory (MRPT),[48,49] configuration interaction (CI),[50–53] coupled cluster (CC),[54–63] multiconfiguration self-consistent field (MCSCF),[64,65] algebraic diagrammatic construction (ADC)[66–70] and G_0W_0[71–74] methods. The majority of these capabilities are available for both molecules and crystalline materials.

1. *Molecular implementations*

The PySCF CI module implements solvers for configuration interaction with single and double excitations (CISD), and a general full configuration interaction (FCI) solver that

```
import pyscf
frag = pyscf.M(atom=’frag.xyz’, basis=’ccpvtz’)
env  = pyscf.M(atom=’env.xyz’, basis=’sto-3g’)
sys = frag + env

def embedding_gradients(sys):
    # Regular HF energy and nuclear gradients of the entire system
    sys_hf = sys.HF().run()
    grad_sys = sys_hf.Gradients().kernel()

    # Construct a CASCI-like effective 1-electron Hamiltonian for the fragment
    # with the presence of outlying atoms in the environment. dm_env is the
    # density matrix in the environment block
    dm_env = sys_hf.make_rdm1()
    dm_env[frag.nao:,:] = dm_env[:,frag.nao:] = 0
    frag_hcore_eff = (sys_hf.get_hcore() + sys_hf.get_veff(sys, dm_env))[:frag.nao, :frag.nao]

    # Customize the zeroth order calculation by overwriting the core Hamiltonian.
    # HF and CISD now provide the embedding wavefunction on fragment.
    geom_frag = sys.atom_coords(unit=’Angstrom’)[:frag.natm]
    frag.set_geom_(geom_frag)
    frag_hf = frag.HF()
    frag_hf.get_hcore = lambda *args: frag_hcore_eff
    frag_hf.run()()
    frag_ci = frag_hf.CISD().run()

    # The .Gradients() method enables a regular analytical nuclear gradient object
    # to evaluate the Hellmann-Feynman forces on fragment using the first order
    # derivatives of the original fragment Hamiltonian and the variational
    # embedding wavefunction.
    grad_hf_frag = frag_hf.Gradients().kernel()
    grad_ci_frag = frag_ci.Gradients().kernel()

    # Approximate the energy and gradients of the entire system with the post-HF
    # correction on fragment
    approx_e = sys_hf.e_tot + frag_ci.e_tot - frag_hf.e_tot
    approx_grad = grad_sys
    approx_grad[:frag.natm] += grad_ci - grad_hf
    print(’Approximate gradients:\n’, approx_grad)
    return approx_e, approx_grad

new_sys = pyscf.geomopt.as_pyscf_method(sys,\
          embedding_gradients).Gradients().optimizer().kernel()
```

FIG. 4: An advanced example that implements geometry optimization based on a WFT-in-HF embedding model using standard PySCF APIs.

can treat fermion, boson and coupled fermion-boson Hamiltonians. The FCI solver is heavily optimized for its multithreaded performance and can efficiently handle active spaces with up to 18 electrons in 18 orbitals.

The CC module implements coupled cluster theory with single and double excitations (CCSD)[56,61] and with the perturbative triples correction [CCSD(T)].[57] Λ-equation solvers are implemented to compute one- and two-particle density matrices, as well as the analytic nuclear gradients for the CCSD and CCSD(T) methods.[55,58,59] PySCF also implements various flavours of equation-of-motion CCSD to compute electron affinities (EA), ionization potentials (IP), neutral excitation energies (EE), and spin-flip excitation energies (SF).[54,60,62,63] Experimental support for beyond doubles corrections to IP and EA via IP-EOM-CCSD*[75,76] and EA-EOM-CCSD* is also available. For very large basis sets, PySCF provides an efficient AO-driven pathway which allows calculations with more than 1500 basis functions. An example of this is shown in Figure 7, where the largest CCSD(T) calculation contains 50 electrons and 1500 basis functions.[77]

Second- and third-order algebraic diagrammatic construction (ADC) methods are also available in PySCF for the calculation of molecular electron affinities and ionization potentials[66–70] [EA/IP-ADC(n), n = 2, 3]. These have a lower cost than EA/IP-EOM-CCSD. The advantage of the ADC methods over EOM-CCSD is that their amplitude equations can be solved in one iteration and the eigenvalue problem is Hermitian, which lowers the cost of computing the EA/IP energies and transition intensities.

The MCSCF module provides complete active space con-

TABLE I: Major features of PySCF as of version 1.7.1.

Methods	Molecules	Solids	Comments
HF	Yes	Yes	$\sim$ 10000 AOs[a]
MP2	Yes	Yes	$\sim$ 1500 MOs[a]
DFT	Yes	Yes	$\sim$ 10000 AOs[a]
TDDFT/TDHF/TDA/CIS	Yes	Yes	$\sim$ 10000 AOs[a]
G_0W_0	Yes	Yes	$\sim$ 1500 MOs[a]
CISD	Yes	Yes[b]	$\sim$ 1500 MOs[a]
FCI	Yes	Yes[b]	$\sim$ (18e, 18o)[a]
IP/EA-ADC(2)	Yes	No	$\sim$ 500 MOs[a,c]
IP/EA-ADC(2)-X	Yes	No	$\sim$ 500 MOs[a,c]
IP/EA-ADC(3)	Yes	No	$\sim$ 500 MOs[a,c]
CCSD	Yes	Yes	$\sim$ 1500 MOs[a]
CCSD(T)	Yes	Yes	$\sim$ 1500 MOs[a]
IP/EA/EE-EOM-CCSD[d]	Yes	Yes	$\sim$ 1500 MOs[a]
MCSCF	Yes	Yes[b]	$\sim$ 3000 AOs,[a] 30–50 active orbitals[e]
MRPT	Yes	Yes[b]	$\sim$ 1500 MOs,[a] 30–50 active orbitals[e]
QM/MM	Yes	No	
Semiempirical	Yes	No	MINDO3
Relativity	Yes	No	ECP and scalar-relativistic corrections for all methods. 2-component methods for HF, DFT, DMRG and SHCI. 4-component methods for HF and DFT.
Gradients	Yes	No	HF, MP2, DFT, TDDFT, CISD, CCSD, CCSD(T), MCSCF and MINDO3
Hessian	Yes	No	HF and DFT
Orbital Localizer	Yes	Yes	NAO, meta-Löwdin, IAO/IBO, VVO/LIVVO, Foster-Boys, Edmiston–Ruedenberg, Pipek–Mezey and Maximally-localized Wannier functions
Properties	Yes	Yes[f]	EFGs, Mössbauer spectroscopy, NMR, magnetizability, and polarizability, *etc.*
Solvation	Yes	No	ddCOSMO, ddPCM, and polarizable embedding
AO, MO integrals	Yes	Yes	1-electron and 2-electron integrals
Density fitting	Yes	Yes	HF, DFT, MP2 and CCSD
Symmetry	Yes	No[g]	D_{2h} and subgroups for molecular HF, MCSCF, and FCI

[a] An estimate based on a single SMP node with 128 GB memory without density fitting;
[b] Γ-point only;
[c] In-core implementation limited by storing two-electron integrals in memory;
[d] Perturbative corrections to IP and EA via IP-EOM-CCSD* and EA-EOM-CCSD* are available for both molecules and crystals;
[e] Using an external DMRG, SHCI, or FCIQMC program (as listed in Section IV B) as the active space solver;
[f] EFGs and Mössbauer spectra only;
[g] Experimental support for point-group and time-reversal symmetries in crystals at the SCF and MP2 levels.

```
import pyscf
mol = pyscf.M(atom = ’N 0 0 0; N 0 0 1.1’ , basis = ’ccpvdz’)
mf = mol.RKS()
mf.xc =’CAMB3LYP’
mf.xc = ’’’0.19*SR_HF(0.33) + 0.65*LR_HF(0.33) + 0.46*ITYH + 0.35*B88, 0.19*VWN5 + 0.81*LYP’’’
mf.xc = ’RSH(0.33, 0.65, -0.46) + 0.46*ITYH + 0.35*B88, 0.19*VWN5 + 0.81*LYP’
e_mf = mf.kernel()
```

FIG. 5: An example of two customized RSH functionals that are equivalent to the CAM-B3LYP functional.

figuration interaction (CASCI) and complete active space self-consistent field (CASSCF)[64,65] methods for multi-reference problems. As discussed in section IV B, the module also provides a general second-order orbital optimizer[78] that can optimize the orbitals of external methods, with native interfaces for the orbital optimization of density matrix renormalization group (DMRG),[29,30] full configuration interaction quantum Monte Carlo (FCIQMC),[35,79] and selected configuration interaction wavefunctions.[31,32] Starting from a CASCI or CASSCF wavefunction, PySCF also implements the strongly-contracted second-order n-electron valence perturbation theory[48,49] (SC-NEVPT2) in the MRPT module to include additional dynamic correlation. Together with external active-space solvers this enables one to treat relatively large active spaces for such calculations, as illustrated in Figure 8.

The Journal of Chemical Physics ACCEPTED MANUSCRIPT

This is the author's peer reviewed, accepted manuscript. However, the online version of record will be different from this version once it has been copyedited and typeset.

PLEASE CITE THIS ARTICLE AS DOI:10.1063/5.0006074

AIP Publishing

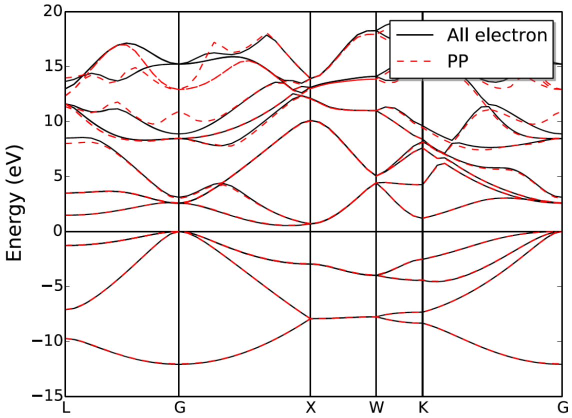


FIG. 6: All-electron and pseudopotential LDA band structures of the Si crystal. Reprinted from Ref. 28, with the permission of AIP Publishing.

2. *Crystalline implementations*

As discussed in section III, the PySCF implementations of many-body methods for crystalline systems closely parallel their molecular implementations. In fact, all molecular modules can be used to carry out calculations in solids at the Γ-point and many modules (those supporting complex integrals) can be used at any other single k-point. Such single k-point calculations only require the appropriate periodic integrals to be supplied to the many-body solver (Figure 9). For those modules that support complex integrals, twist averaging can then be performed to sample the Brillouin zone. To use savings from k-point symmetries, an additional summation over momentum conserving k-point contributions needs to be explicitly implemented. Such implementations are provided for MP2, CCSD, CCSD(T), IP/EA-EOM-CCSD[27] and EE-EOM-CCSD,[80] and G_0W_0. For example, Figure 10 shows the MP2 correlation energy and the CIS excitation energy of MgO, calculated using periodic density-fitted implementations; the largest system shown, with a $7 \times 7 \times 7$ k-point mesh, correlates 5,488 valence electrons in 9,261 orbitals. Furthermore, Figure 11 shows some examples of periodic correlated calculations on NiO carried out using the G_0W_0 and CCSD methods.

C. Efficiency

In Table I, we provide rough estimates of the sizes of problems that can be tackled using PySCF for each electronic structure method. Figs. 7, 8, 10, and 11 illustrate some real-world examples of calculations performed using PySCF. Note that the size of system that can be treated is a function of the computational resources available; the estimates given above assume relatively standard and modest computational resources, e.g. a node of a cluster, or a few dozen cores. For more details of the runtime environment and program settings for similar performance benchmarks, we refer readers to the benchmark page of the PySCF website `www.pyscf.org`. The implementation and performance of PySCF on massively parallel architectures is discussed in section V H.

TABLE II: Wall time (in seconds) for building the Fock matrix in a supercell DFT calculation of water clusters with the GTH-TZV2P[26,47] basis set, using the SVWN[82] and the PBE[83] XC functionals and corresponding pseudopotenials, respectively. Integral screening and lattice summation cutoff were controlled by an overall threshold of 10^{-6} a.u. for Fock matrix elements. The calculations were performed on one computer node with 32 Intel Xeon Broadwell (E5-2697v4) processors.

System	N_{AO}[a]	SVWN	PBE
$[H_2O]_{32}$	1,280	8	23
$[H_2O]_{64}$	2,560	20	56
$[H_2O]_{128}$	5,120	74	253
$[H_2O]_{256}$	12,800	276	1201
$[H_2O]_{512}$	25,600	1279	4823

[a] Number of AO basis functions.

For molecular calculations using mean-field methods, PySCF can treat systems with more than 10,000 AO basis functions without difficulty. Fig. 15 shows the time of building the HF Fock matrix for a large water cluster with more than 12,000 basis functions. With the integral screening threshold set to 10^{-13} a.u., it takes only around 7 hours on one computer node with 32 CPU cores. Applying MPI parallelization further reduces the Fock-build time (see section V H). For periodic boundary calculations at the DFT level using pure XC functionals, even larger systems can be treated using pseudopotentials and a multi-grid implementation. Table II presents an example of such a calculation, where for the largest system considered ($[H_2O]_{512}$ with more than 25,000 basis functions), the Fock-build time is about an hour or less on a single node.

To demonstrate the efficiency of the many-body method implementations, in Tables III and IV we show timing data of exemplary CCSD and FCI calculations. It is clear that systems with more than 1,500 basis functions can be easily treated at the CCSD level and that the FCI implementation in PySCF is very efficient. In a similar way, the estimated performance for other many-body methods implemented in PySCF is listed in Table I.

D. Properties

At the mean-field level, the current PySCF program can compute various nonrelativistic and four-component relativistic molecular properties. These include NMR shielding and spin-spin coupling tensors,[84–89] electronic g-tensors,[90–93] nuclear spin-rotation constants and rotational g-tensors,[94,95] hyperfine coupling (HFC) tensors,[96,97] electron spin-rotation (ESR) tensors,[98,99] magnetizability tensors,[94,100,101] zero-field splitting (ZFS) tensors,[102–104] as well as static and dynamic polarizability and hyper-polarizability tensors. The contributions from spin-orbit coupling and spin-spin coupling can

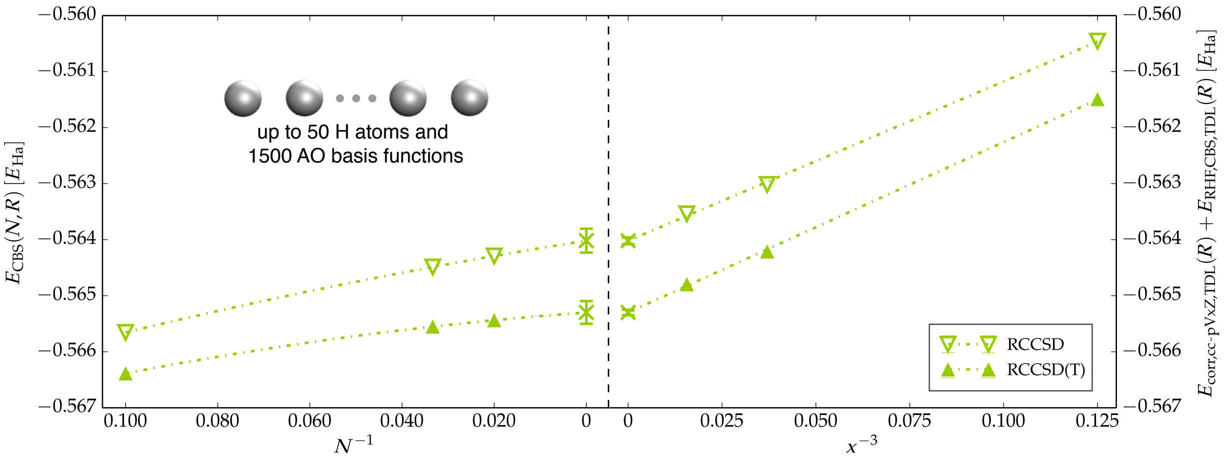


FIG. 7: Energies of a hydrogen chain computed at the restricted CCSD and CCSD(T) levels extrapolated to the complete basis set (CBS) and thermodynamic limits. The left-hand panel shows extrapolation of $E_{\mathrm{CBS}}(N)$ versus $1/N$, where N is the number of atoms; while the right-hand panel shows extrapolation of $E_{\mathrm{cc-pVxZ}}(N \to \infty)$ versus $1/x^3$ with x equal to 2, 3 and 4 corresponding to double-, triple- and quadruple-zeta basis, respectively. Adapted from Ref. 77.

TABLE III: Wall time (in seconds) for the first CCSD iteration in AO-driven CCSD calculations on hydrogen chains. The threshold of integral screening was set to 10^{-13} a.u.. For these hydrogen chain molecules, CCSD takes around 10 iterations to converge. The calculations were performed on one computer node with 28 Intel Xeon Broadwell (E5-2697v4) processors.

System/basis set	N_{occ}[a]	N_{virt}[b]	time
H_{30}/cc-pVQZ	15	884	621
H_{30}/cc-pV5Z	15	1631	6887
H_{50}/cc-pVQZ	25	1472	8355

[a] Number of active occupied orbitals;
[b] Number of active virtual orbitals.

TABLE IV: Wall time (in seconds) for one FCI iteration for different active-space sizes. The calculations were performed on one computer node with 32 Intel Xeon Broadwell (E5-2697v4) processors.

Active space	time
(12e, 12o)	0.1
(14e, 14o)	0.7
(16e, 16o)	8
(18e, 18o)	156

also be calculated and included in the g-tensors, HFC tensors, ZFS tensors, and ESR tensors. In magnetic property calculations, approximate gauge-origin invariance is ensured for NMR shielding, g-tensors, and magnetizability tensors via the use of gauge including atomic orbitals.[94,100,101,105,106]

Electric field gradients (EFGs) and Mössbauer parameters[107–109] can be computed using either the mean-field electron density, or the correlated density obtained from non-relativistic Hamiltonians, spin-free exact-two-component (X2C) relativistic Hamiltonians[110–114] or four-component methods, in both molecules and crystals.

Finally, analytic nuclear gradients for the molecular ground state are available at the mean-field level and for many of the electron correlation methods such as MP2, CCSD, CISD, CASCI and CASSCF (see Table I). The CASCI gradient implementation supports the use of external solvers, such as DMRG, and provides gradients for such methods. PySCF also implements the analytical gradients of time-dependent density functional theory (TDDFT) with or without the Tamm-Dancoff approximation (TDA) for excited state geometry optimization. The spin-free X2C relativistic Hamiltonian, frozen core approximations, solvent effects, and molecular mechanics (MM) environments can be combined with any of the nuclear gradient methods. Vibrational frequency and thermochemical analysis can also be performed, using the analytical Hessians from mean-field level calculations, or numerical Hessians of methods based on numerical differentiation of analytical gradients.

E. Orbital localization

PySCF provides two kinds of orbital localization in the LO module. The first kind localizes orbitals based on the atomic character of the basis functions, and can generate intrinsic atomic orbitals (IAOs),[115] natural atomic orbitals (NAOs),[116] and meta-Löwdin orbitals.[117] These AO-based local orbitals can be used to carry out reliable population analysis in arbitrary basis sets.

The second kind optimizes a cost function to produce localized orbitals. PySCF implements Boys localization,[118] Edmiston-Ruedenberg localization,[119] and Pipek–Mezey localization.[120] Starting from the IAOs, one can also use orbital localization based on the Pipek-Mezey procedure to construct the intrinsic bond orbitals (IBOs).[115] A similar method

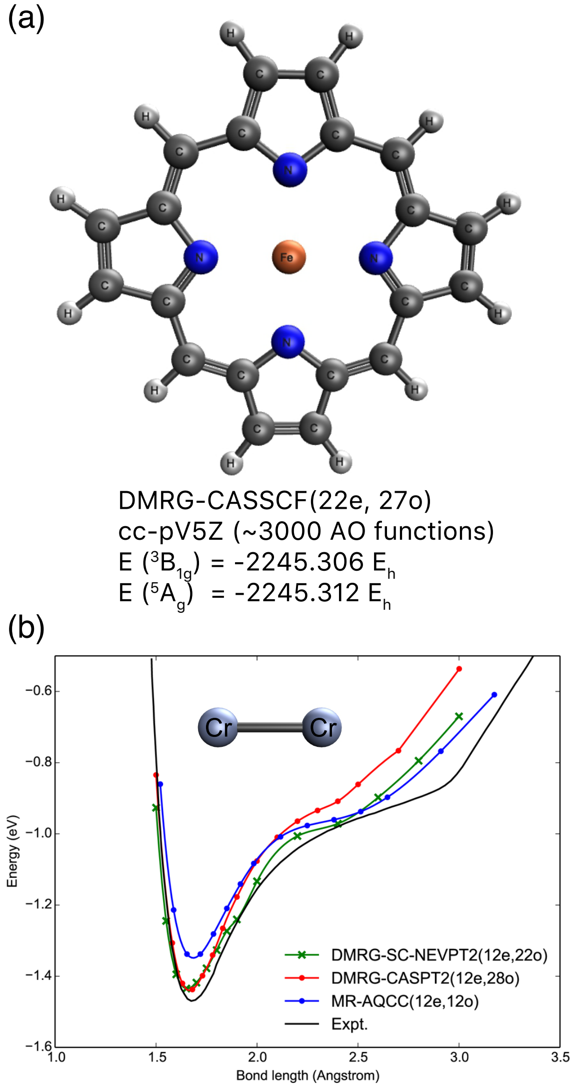


FIG. 8: (a) Ground-state energy calculations for Fe(II)-porphine at the DMRG-CASSCF/cc-pV5Z level with an active space of 22 electrons in 27 orbitals.[78] (b) Potential energy curve for Cr_2 at the DMRG-SC-NEVPT2 (12e, 22o) level, compared to the results from other methods. Adapted with permission from Ref. 49. Copyright (2016) American Chemical Society.

can also be used to construct localized intrinsic valence virtual orbitals that can be used to assign core-excited states.[121] The optimization in these localization routines takes advantage of the second order coiterative augmented Hessian (CIAH) algorithm[122] for rapid convergence.

For crystalline calculations with *k*-point sampling, PySCF also provides maximally-localised Wannier functions (MLWFs) via a native interface to the Wannier90 program.[123] Different types of orbitals are available as initial guesses for the MLWFs, including the atomic orbitals provided by Wannier90, meta-Löwdin orbitals,[117] and localized orbitals from the selected columns of density matrix (SCDM) method.[124,125] Figure 12 illustrates the IBOs and MLWFs of diamond computed by PySCF.

F. QM/MM and solvent

PySCF incorporates two continuum solvation models, namely, the conductor-like screening model[126] (COSMO) and the polarizable continuum model using the integral equation formalism[127,128] (IEF-PCM). Both of them are implemented efficiently via a domain decomposition (dd) approach,[129–133] and are compatible with most of the electronic structure methods in PySCF. Furthermore, besides equilibrium solvation where the solvent polarization is governed by the static electric susceptibility, non-equilibrium solvation can also be treated within the framework of TDDFT, in order to describe fast solvent response with respect to abrupt changes of the solute charge density. As an example, in Ref. 134, the COSMO method was used to mimic the protein environment of nitrogenase in electronic structure calculations for the P-cluster (Figure 13). For excited states generated by TDA, the polarizable embedding model[135] can also be used through an interface to the external library CPPE.[135,136]

Currently, PySCF provides some limited functionality for performing QM/MM calculations by adding classical point charges to the QM region. The implementation supports all molecular electronic structure methods by decorating the underlying SCF methods. In addition, MM charges can be used together with the X2C method and implicit solvent treatments.

G. Relativistic treatments

PySCF provides several ways to include relativistic effects. In the framework of scalar Hamiltonians, spin-free X2C theory,[137] scalar effective core potentials[138] (ECP) and relativistic pseudo-potentials can all be used for all methods in calculations of the energy, nuclear gradients and nuclear Hessians. At the next level of relativistic approximations, PySCF provides spin-orbit ECP integrals, and one-body and two-body spin-orbit interactions from the Breit-Pauli Hamiltonian and X2C Hamiltonian for the spin-orbit coupling effects.[139] Two component Hamiltonians with the X2C one-electron approximation, and four-component Dirac-Coulomb, Dirac-Coulomb-Gaunt, and Dirac-Coulomb-Breit Hamiltonians are all supported in mean-field molecular calculations.

H. MPI implementations

In PySCF, distributed parallelism with MPI is implemented via an extension to the PySCF main library known as MPI4PySCF. The current MPI extension supports the most common methods in quantum chemistry and crystalline material computations. Table V lists the available MPI-parallel alternatives to the default serial (OpenMP) implementations. The MPI-enabled modules implement almost identical APIs

```python
import pyscf
cell = pyscf.M(atom=..., a=...) # 'a' defines lattice vectors
mf = cell.HF(kpt = [0.23,0.23,0.23]).run()
# Use PBC CCSD class so integrals are handled
# correctly with respect to Coulomb divergences
mycc = pyscf.pbc.cc.CCSD(mf)
# molecular CCSD code used to compute correlation energy at single k-point
converged, ecorr = pyscf.cc.ccsd.kernel(mycc)
```

FIG. 9: Illustration of using the molecular code to compute an energy in crystal at a single *k*-point.

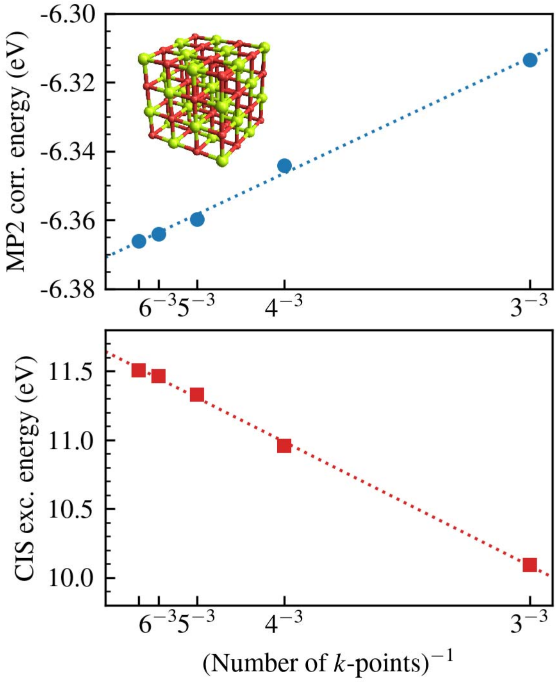


FIG. 10: Periodic MP2 correlation energy per unit cell (top) and CIS excitation energy (bottom) as a function of the number of *k*-points sampled in the Brillouin zone for the MgO crystal.

TABLE V: Methods with MPI support. For solids, MPI support is currently provided only at the level of parallelization over *k*-points.

Methods	Molecules	Solids
HF	Yes	Yes
DFT	Yes	Yes
MP2	Yes[a]	Yes
CCSD	Yes[a]	Yes

[a] closed shell systems only

to the serial ones, allowing the same script to be used for serial jobs and MPI-parallel jobs (Figure 14). The efficiency of the MPI implementation is demonstrated in Figure 15, which shows the wall time and speedup of Fock builds for a system with 12,288 AOs with up to 64 MPI processes, each with 32 OpenMP threads.

To retain the simplicity of the PySCF package structure, we use a server-client mechanism to execute the MPI parallel code. In particular, we use MPI to start the Python interpreter as a daemon that receives both the functions and data on remote nodes. When a parallel session is activated, the master process sends the functions and data to the daemons. The function object is decoded remotely and then executed. For example, when building the Fock matrix in the PySCF MPI implementation, the Fock-build function running on the master process first sends itself to the Python interpreters running on the clients. After the function is decoded on the clients, input variables (like the density matrix) are distributed by the master process through MPI. Each client evaluates a subset of the four-center two-electron integrals (with load balancing performed among the clients) and constructs a partial Fock matrix, similarly to the Fock-build functions in other MPI implementations. After sending the partial Fock matrices back to the master process, the client suspends itself until it receives the next function. The master process assembles the Fock matrices and then moves on to the next part of the code. The above strategy is quite different from traditional MPI programs that hard-code MPI functionality into the code and initiate the MPI parallel context at the beginning of the program. This PySCF design brings the important benefit of being able to switch on and off MPI parallelism freely in the program without the need to be aware of the MPI-parallel context. See Ref. 1 for a more detailed discussion of PySCF MPI mode innovations.

VI. THE PYSCF SIMULATION ECOSYSTEM

PySCF is widely used as a development tool, and many groups have developed and made available their own projects that either interface to PySCF or can be used in a tightly coupled manner to access greater functionality. We provide a few examples of the growing PySCF ecosystem below, which we separate into use cases: (1) external projects to which PySCF provides and maintains a native interface, and (2) external projects that build on PySCF.

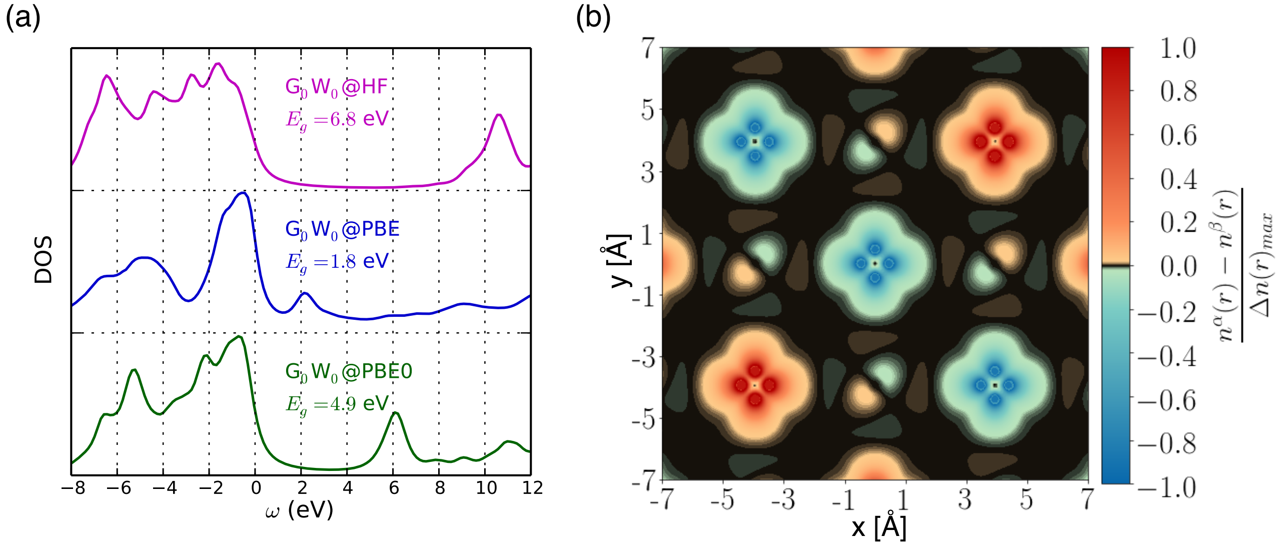


FIG. 11: Electronic structure calculations for antiferromagnetic NiO. (a) Density of states and band gaps computed by G_0W_0. (b) Normalized spin density on the (100) surface by CCSD (the Ni atom is located at the center). Adapted from Ref. 81.

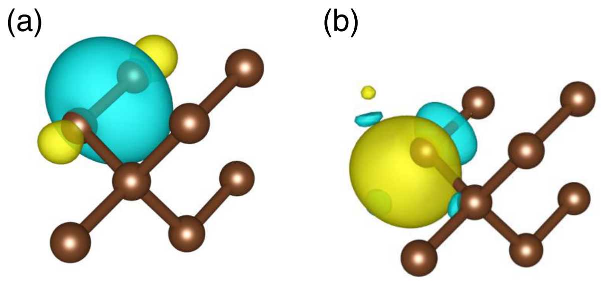


FIG. 12: (a) IBOs for diamond at the Γ-point (showing one σ bond); (b) MLWFs for diamond computed within the valence IAO subspace (showing one sp^3 orbital).

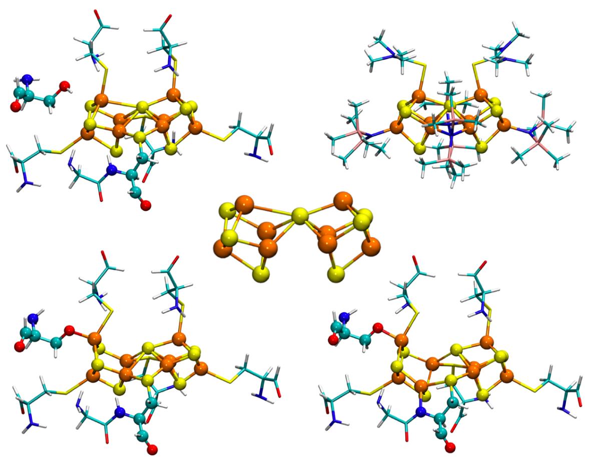


FIG. 13: Illustration of P-cluster (the $[Fe_8S_7]$ cluster of nitrogenase) calculations where the COSMO solvation model was used to mimic the protein environment of nitrogenase beyond the first coordination sphere. Fe, orange; S, yellow; C, cyan; O, red; N, blue; H, white; Si, pink. Adapted from Ref. 134.

A. External projects with native interfaces

PySCF currently maintains a few native interfaces to external projects, including:

- GEOMETRIC[140] and PYBERNY.[141] These two libraries provide the capability to perform geometry optimization and interfaces to them are provided in the PySCF GEOMOPT module. As shown in Figure 4, given a method that provides energies and nuclear gradients, the geometry optimization module generates an object that can then be used by these external optimization libraries.
- DFTD3.[142,143] This interface allows to add the DFTD3[142] correction to the total ground state energy as well as to the nuclear gradients in geometry optimizations.
- DMRG, SHCI, and FCIQMC programs (BLOCK,[29] CHEMPS2,[30] DICE,[31–33] ARROW,[31,33,34] and NECI[35]). These interfaces closely follow the conventions of PySCF's FCI module. As such, they can be used to replace the FCI solver in MCSCF methods (CASCI and CASSCF) to study large active space multi-reference problems.
- LIBXC[39] and XCFUN.[40] These two libraries are tightly integrated into the PySCF code. While the PySCF DFT module allows the user to customize exchange correlation (XC) functionals by linearly combining differ-

```
# run in cmdline:
# mpirun -np 4 python input.py

import pyscf
mol = pyscf.M(...)

# Serial task
from pyscf import dft
mf = dft.RKS(mol).run(xc='b3lyp')
J, K = mf.get_jk(mol, mf.make_rdm1())

# MPI-parallel task
from mpi4pyscf import dft
mf = dft.RKS(mol).run(xc='b3lyp')
J, K = mf.get_jk(mol, mf.make_rdm1())
```

FIG. 14: Code snippet showing the similarity between serial and MPI-parallel DFT calculations.

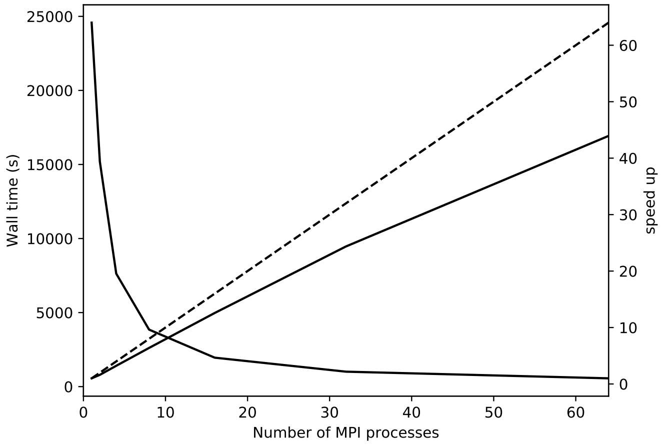


FIG. 15: Computation wall time of building the Fock matrix for the $[H_2O]_{512}$ cluster at the HF/VDZ level (12288 AO functions) using PySCF's MPI implementation. Each MPI process contains 32 OpenMP threads and the speedup is compared to the single-node calculation with 32 OpenMP threads.

ent functionals, the individual XC functionals and their derivatives are evaluated within these libraries.

- TBLIS.[144–146] The tensor contraction library TBLIS offers similar functionality to the numpy.einsum function while delivering substantial speedups. Unlike the BLAS-based "transpose-GEMM-transpose" scheme which involves a high memory footprint due to the transposed tensor intermediates, TBLIS achieves optimal tensor contraction performance without such memory overhead. The TBLIS interface in PySCF provides an einsum function which implements the numpy.einsum API but with the TBLIS library as the contraction back-end.

- CPPE.[135,136] This library provides a polarizable embedding solvent model and can be integrated into PySCF calculations for ground-state mean-field and post-SCF methods. In addition, an interface to TDA is currently supported for excited-state calculations.

B. External projects that build on PySCF

There are many examples in the literature of quantum chemistry and electronic structure simulation packages that build on PySCF. The list below is by no means exhaustive, but gives an idea of the range of projects using PySCF today.

1. *Quantum Monte Carlo*. Several quantum Monte Carlo programs, such as QMCPACK,[147] pyQMC,[148] QWalk,[149] and HANDE[150] support reading wavefunctions and/or Hamiltonians generated by PySCF. In the case of pyQMC, PySCF is integrated as a dependent module.

2. *Quantum embedding packages*. Many flavours of quantum embedding, including density matrix embedding and dynamical mean-field theory, have been implemented on top of PySCF. Examples of such packages include QSoME,[151–153] pDMET,[154,155] PyDMFET,[156] Potato,[157,158] and the OpenQEMIST package,[16] which all use PySCF to manipulate wavefunctions and embedding Hamiltonians and to provide many-electron solvers.

3. *General quantum chemistry*. PySCF can be found as a component of tools developed for many different kinds of calculations, including localized active space self-consistent field (LASSCF),[154] multiconfiguration pair-density functional theory (MC-PDFT),[159] and state-averaged CASSCF energy and analytical gradient evaluation (these all use the PySCF MCSCF module to optimize multi-reference wavefunctions), as well as for localized orbital construction via the PyWannier90 library.[155] The PyMBE package,[160] which implements the many-body expanded full CI method,[161–164] utilizes PySCF to perform all the underlying electronic structure calculations. Green's functions methods such as the second-order Green's function theory (GF2) and the self-consistent GW approximation have been explored using PySCF as the underlying ab initio infrastructure.[165] In the linear scaling program LSQC,[166,167] PySCF is used to generate reference wavefunctions and integrals for the cluster-in-molecule local correlation method. The APDFT (alchemical perturbation density functional theory) program[168,169] interfaces to PySCF for QM calculations. In the PySCF-NAO project,[170] large-scale ground-state and excited-state methods are implemented based on additional support for numerical atomic orbitals, which has been integrated into an active branch of PySCF.

The Journal of Chemical Physics ACCEPTED MANUSCRIPT
This is the author's peer reviewed, accepted manuscript. However, the online version of record will be different from this version once it has been copyedited and typeset.
PLEASE CITE THIS ARTICLE AS DOI:10.1063/5.0006074
AIP Publishing

The PYFLOSIC package[171] evaluates self-interaction corrections with Fermi-Löwdin orbitals in conjunction with the PYSCF DFT module. Further, PYSCF FCI capabilities are used in the MOLSTURM package[172] for the development of Coulomb Sturmian basis functions, and PYSCF post-HF methods appear in VELOXCHEM[173] and ADCC[174] for spectroscopic and excited-state simulations.

VII. BEYOND ELECTRONIC STRUCTURE

A. PySCF in the materials genome initiative and machine learning

As discussed in section I, one of our objectives when developing PYSCF was to create a tool which could be used by non-specialist researchers in other fields. With the integration of machine learning techniques into molecular and materials simulations, we find that PYSCF is being used in many applications in conjunction with machine learning. For example, the flexibility of the PYSCF DFT module has allowed it to be used to test exchange-correlation functionals generated by machine-learning protocols in several projects,[7,8] and has been integrated into other machine learning workflows.[9,10] PYSCF can be used as a large-scale computational engine for quantum chemistry data generation.[5,6] Also, in the context of machine learning of wavefunctions, PYSCF has been used as the starting point to develop neural network based approaches for SCF initial guesses,[11] for the learning of HF orbitals by the DeepMind team,[12] and for Hamiltonian integrals used by fermionic neural nets in NETKET.[13]

B. PySCF in quantum information science

Another area where PYSCF has been rapidly adopted as a development tool is in the area of quantum information science and quantum computing. This is likely because Python is the de-facto standard programming language in the quantum computing community. For example, PYSCF is one of the standard prerequisites to carry out molecular simulations in the OPENFERMION[14] library, the QISKIT-AQUA[15] library and the OPENQEMIST[16] package. Via PYSCF's GitHub page, we see a rapidly increasing number of quantum information projects which include PYSCF as a program dependency.

VIII. OUTLOOK

After five years of development, the PYSCF project can probably now be considered to be a feature complete and mature tool. Although no single package can be optimal for all tasks, we believe PYSCF to a large extent meets its original development criteria of forming a library that is not only useful in simulations but also in enabling the customization and development of new electronic structure methods.

With the recent release of version 1.7, the current year marks the end of development of the version 1 branch of PYSCF. As we look towards PYSCF version 2, we expect to build additional innovations, for example, in the areas of faster electronic structure methods for very large systems, further support and integration for machine learning and quantum computing applications, better integration of high-performance computing libraries and more parallel implementations, and perhaps even forays into dynamics and classical simulations. Beyond feature development, we will expand our efforts in documentation and in quality assurance and testing. We expect the directions of implementation to continue to be guided by and organically grow out of the established PYSCF ecosystem. However, regardless of the scientific directions and methods implemented within PYSCF, the guiding philosophy described in this article will continue to lie at the heart of PYSCF's development. We believe these guiding principles will help ensure that PYSCF remains a powerful and useful tool in the community for many years to come.

DATA AVAILABILITY STATEMENT

The data that supports the findings of this study are available within the article, and/or from the corresponding author upon reasonable request.

ACKNOWLEDGMENTS

As a large package, the development of PYSCF has been supported by different sources. Support from the US National Science Foundation via award no. 1931258 (T.C.B., G.K-L.C., and L.K.W.) is acknowledged to integrate high-performance parallel infrastructure and faster mean-field methods into PYSCF. Support from the US National Science Foundation via award no. 1657286 (G.K.-L.C.) and award no. 1848369 (T.C.B.) is acknowledged for various aspects of the development of many-electron wavefunction methods with periodic boundary conditions. Support for integrating PYSCF into quantum computing platforms is provided partially by the Department of Energy via award no. 19374 (G.K.-L.C). The Simons Foundation is gratefully acknowledged for providing additional support for the continued maintenance and development of PYSCF. The Flatiron Institute is a division of the Simons Foundation. M.B. acknowledges support from the Departemento de Educación of the Basque Government through a PhD grant, as well as from Euskampus and the DIPC at the initial stages of his work. J.C. is supported by the Center for Molecular Magnetic Quantum Materials (M2QM), an Energy Frontier Research Center funded by the US Department of Energy, Office of Science, Basic Energy Sciences under Award DE-SC0019330. J.J.E. acknowledges financial support from the Alexander von Humboldt Foundation and the Independent Research Fund Denmark. M.R.H. and H.Q.P. were partially supported by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, Division of Chemical Sciences, Geosciences and Biosciences under Award #DE-FG02-17ER16362, while working in the group of Laura Gagliardi at the University of Minnesota. P.K. acknowledges financial support from the Fellows Gipuzkoa program of the Gipuzkoako Foru Aldundia through the FEDER funding scheme of the European Union. S.L. has been supported by the Academy of Finland (Suomen Akatemia) through project number 311149. A.P. thanks Swiss NSF for the support provided through the Early Postdoc. Mobility program (project P2ELP2_175281). H.F.S acknowledges the financial support from the European Union via Marie Skłodowska-Curie Grant Agreement No. 754388 and LMUexcellent within the German Excellence Initiative (No. ZUK22). S.B. and J.E.T.S. gratefully acknowledge support from a fellowship through The Molecular Sciences Software Institute under NSF Grant ACI-1547580. S.S. acknowledges support of NSF grant CHE-1800584. S.U. acknowledges the support of NSF grant CHE-1762337. The National Science Foundation Graduate Research Fellowship Program is acknowledged for support of J.M.Y.

[1] Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma, S. Wouters, and G. K.-L. Chan, Wiley Interdiscip. Rev.: Comput. Mol. Sci. **8**, e1340 (2018).
[2] "Python-based Simulations of Chemistry Framework," (2020), https://github.com/pyscf/pyscf (Accessed 16 Apr 2020).
[3] "PySCF: Python-based Simulations of Chemistry Framework," (2020), https://pypi.org/project/pyscf (Accessed 16 Apr 2020).
[4] "Python-based Simulations of Chemistry Framework," (2020), https://anaconda.org/pyscf/pyscf (Accessed 16 Apr 2020).
[5] G. Chen, P. Chen, C.-Y. Hsieh, C.-K. Lee, B. Liao, R. Liao, W. Liu, J. Qiu, Q. Sun, J. Tang, R. Zemel, and S. Zhang, (2019), arXiv:1906.09427 [cs.LG].
[6] C. Lu, Q. Liu, Q. Sun, C.-Y. Hsieh, S. Zhang, L. Shi, and C.-K. Lee, (2019), arXiv:1910.13551 [physics.chem-ph].
[7] S. Dick and M. Fernandez-Serra, (2019), 10.26434/chemrxiv.9947312, DOI: 10.26434/chemrxiv.9947312.
[8] H. Ji and Y. Jung, J. Chem. Phys. **148**, 241742 (2018).
[9] J. Hermann, Z. Schätzle, and F. Noé, (2019), arXiv:1909.08423 [physics.comp-ph].
[10] J. Han, L. Zhang, and W. E, J. Comput. Phys. **399**, 108929 (2019).
[11] J. Cartus, https://github.com/jcartus/SCFInitialGuess (Accessed 21 Feb 2020).
[12] D. Pfau, J. S. Spencer, A. G. d. G. Matthews, and W. M. C. Foulkes, (2019), arXiv:1909.02487 [physics.chem-ph], arXiv:1909.02487.
[13] K. Choo, A. Mezzacapo, and G. Carleo, (2019), arXiv:1909.12852 [physics.comp-ph], arXiv:1909.12852.
[14] J. R. McClean, K. J. Sung, I. D. Kivlichan, Y. Cao, C. Dai, E. S. Fried, C. Gidney, B. Gimby, P. Gokhale, T. Häner, T. Hardikar, V. Havlíček, O. Higgott, C. Huang, J. Izaac, Z. Jiang, X. Liu, S. McArdle, M. Neeley, T. O'Brien, B. O'Gorman, I. Ozfidan, M. D. Radin, J. Romero, N. Rubin, N. P. D. Sawaya, K. Setia, S. Sim, D. S. Steiger, M. Steudtner, Q. Sun, W. Sun, D. Wang, F. Zhang, and R. Babbush, (2017), arXiv:1710.07629 [quant-ph].
[15] H. Abraham, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, G. Alexandrowics, E. Arbel, A. Asfaw, C. Azaustre, AzizNgoueya, P. Barkoutsos, G. Barron, L. Bello, Y. Ben-Haim, D. Bevenius, L. S. Bishop, S. Bosch, S. Bravyi, D. Bucher, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Chen, C.-F. Chen, R. Chen, J. M. Chow, C. Claus, C. Clauss, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, C. Culver, A. D. Córcoles-Gonzales, S. Dague, T. E. Dandachi, M. Dartiailh, DavideFrr, A. R. Davila, D. Ding, J. Doi, E. Drechsler, Drew, E. Dumitrescu, K. Dumon, I. Duran, K. EL-Safty, E. Eastman, P. Eendebak, D. Egger, M. Everitt, P. M. Fernández, A. H. Ferrera, A. Frisch, A. Fuhrer, M. GEORGE, J. Gacon, Gadi, B. G. Gago, J. M. Gambetta, A. Gammanpila, L. Garcia, S. Garion, J. Gomez-Mosquera, S. de la Puente González, I. Gould, D. Greenberg, D. Grinko, W. Guan, J. A. Gunnels, I. Haide, I. Hamamura, V. Havlicek, J. Hellmers, Ł. Herok, S. Hillmich, H. Horii, C. Howington, S. Hu, W. Hu, H. Imai, T. Imamichi, K. Ishizaki, R. Iten, T. Itoko, A. Javadi-Abhari, Jessica, K. Johns, T. Kachmann, N. Kanazawa, Kang-Bae, A. Karazeev, P. Kassebaum, S. King, Knabberjoe, A. Kovyrshin, V. Krishnan, K. Krsulich, G. Kus, R. LaRose, R. Lambert, J. Latone, S. Lawrence, D. Liu, P. Liu, Y. Maeng, A. Malyshev, J. Marecek, M. Marques, D. Mathews, A. Matsuo, D. T. McClure, C. McGarry, D. McKay, S. Meesala, M. Mevissen, A. Mezzacapo, R. Midha, Z. Minev, N. Moll, M. D. Mooring, R. Morales, N. Moran, P. Murali, J. Müggenburg, D. Nadlinger, G. Nannicini, P. Nation, Y. Naveh, P. Neuweiler, P. Niroula, H. Norlen, L. J. O'Riordan, O. Ogunbayo, P. Ollitrault, S. Oud, D. Padilha, H. Paik, S. Perriello, A. Phan, M. Pistoia, A. Pozas-iKerstjens, V. Prutyanov, D. Puzzuoli, J. Pérez, Quintiii, R. Raymond, R. M.-C. Redondo, M. Reuter, J. Rice, D. M. Rodríguez, M. Rossmannek, M. Ryu, T. SAPV, SamFerracin, M. Sandberg, N. Sathaye, B. Schmitt, C. Schnabel, Z. Schoenfeld, T. L. Scholten, E. Schoute, I. F. Sertage, K. Setia, N. Shammah, Y. Shi, A. Silva, A. Simonetto, N. Singstock, Y. Siraichi, I. Sitdikov, S. Sivarajah, M. B. Sletfjerding, J. A. Smolin, M. Soeken, I. O. Sokolov, D. Steenken, M. Stypulkoski, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, S. Thomas, M. Tillet, M. Tod, E. de la Torre, K. Trabing, M. Treinish, TrishaPe, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, A. C. Vazquez, D. Vogt-Lee, C. Vuillot, J. Weaver, R. Wieczorek, J. A. Wildstrom, R. Wille, E. Winston, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, J. Wootton, D. Yeralin, R. Young, J. Yu, C. Zachow, L. Zdanski, C. Zoufal, Zoufalc, azulehner, bcamorrison, brandhsn, chlorophyll zz, dime10, drholmie, elfrocampeador, faisaldebouni, fanizzamarco, gruu, kanejess, klinvill, kurarrr, lerongil, ma5x, merav aharoni, ordmoj, sethmerkel, strickroman, sumitpuri, tigerjack, toural, vvilpas, willhbang, yang.luh, and yotamvakninibm, "Qiskit: An open-source framework for quantum computing," (2019), DOI: 10.5281/zenodo.2562110.
[16] T. Yamazaki, S. Matsuura, A. Narimani, A. Saidmuradov, and A. Zaribafiyan, (2018), arXiv:1806.01305 [quant-ph].
[17] "PySCF: the Python-based Simulations of Chemistry Framework," (2020), http://pyscf.org (Accessed 16 Apr 2020).
[18] M. Valiev, E. Bylaska, N. Govind, K. Kowalski, T. Straatsma, H. Van Dam, D. Wang, J. Nieplocha, E. Apra, T. Windus, and W. de Jong, Comput. Phys. Commun. **181**, 1477 (2010).
[19] F. Furche, R. Ahlrichs, C. Hättig, W. Klopper, M. Sierka, and F. Weigend, Wiley Interdiscip. Rev.: Comput. Mol. Sci. **4**, 91 (2014).
[20] Y. Shao, Z. Gan, E. Epifanovsky, A. T. Gilbert, M. Wormit, J. Kussmann, A. W. Lange, A. Behn, J. Deng, X. Feng, D. Ghosh, M. Goldey, P. R. Horn, L. D. Jacobson, I. Kaliman, R. Z. Khaliullin, T. Kuś, A. Landau, J. Liu, E. I. Proynov, Y. M. Rhee, R. M. Richard, M. A. Rohrdanz, R. P. Steele, E. J. Sundstrom, H. L. Woodcock, P. M. Zimmerman, D. Zuev, B. Albrecht, E. Alguire, B. Austin, G. J. O. Beran, Y. A. Bernard, E. Berquist, K. Brandhorst, K. B. Bravaya, S. T. Brown, D. Casanova, C.-M. Chang, Y. Chen, S. H. Chien, K. D. Closser, D. L. Crittenden, M. Diedenhofen, R. A. DiStasio, H. Do, A. D. Dutoi, R. G. Edgar, S. Fatehi, L. Fusti-Molnar, A. Ghysels, A. Golubeva-Zadorozhnaya, J. Gomes, M. W. Hanson-Heine, P. H. Harbach, A. W. Hauser, E. G. Hohenstein, Z. C. Holden, T.-C. Jagau, H. Ji, B. Kaduk, K. Khistyaev, J. Kim, J. Kim, R. A. King, P. Klunzinger, D. Kosenkov, T. Kowalczyk, C. M. Krauter, K. U. Lao, A. D. Laurent, K. V. Lawler, S. V. Levchenko, C. Y. Lin, F. Liu, E. Livshits, R. C. Lochan, A. Luenser, P. Manohar, S. F. Manzer, S.-P. Mao, N. Mardirossian, A. V. Marenich, S. A. Maurer, N. J. Mayhall, E. Neuscamman, C. M. Oana, R. Olivares-Amaya, D. P. O'Neill, J. A. Parkhill, T. M. Perrine, R. Peverati, A. Prociuk, D. R. Rehn, E. Rosta, N. J. Russ, S. M. Sharada, S. Sharma, D. W. Small, A. Sodt, T. Stein, D. Stück, Y.-C. Su, A. J. Thom, T. Tsuchimochi, V. Vanovschi, L. Vogt, O. Vydrov, T. Wang, M. A. Watson, J. Wenzel, A. White, C. F. Williams, J. Yang, S. Yeganeh, S. R. Yost, Z.-Q. You, I. Y. Zhang, X. Zhang, Y. Zhao, B. R. Brooks, G. K. Chan, D. M. Chipman, C. J. Cramer, W. A. Goddard, M. S. Gordon, W. J. Hehre, A. Klamt, H. F. Schaefer, M. W. Schmidt, C. D. Sherrill, D. G. Truhlar, A. Warshel, X. Xu, A. Aspuru-Guzik, R. Baer, A. T. Bell, N. A. Besley, J.-D. Chai, A. Dreuw, B. D. Dunietz, T. R. Furlani, S. R. Gwaltney, C.-P. Hsu, Y. Jung, J. Kong, D. S. Lambrecht, W. Liang, C. Ochsenfeld, V. A. Rassolov, L. V. Slipchenko, J. E. Subotnik, T. Van Voorhis, J. M. Herbert, A. I. Krylov, P. M. Gill, and M. Head-Gordon, Mol. Phys. **113**, 184 (2015).
[21] R. M. Parrish, L. A. Burns, D. G. Smith, A. C. Simmonett, A. E. DePrince, E. G. Hohenstein, U. Bozkaya, A. Y. Sokolov, R. Di Remigio,

The Journal of Chemical Physics ACCEPTED MANUSCRIPT

This is the author's peer reviewed, accepted manuscript. However, the online version of record will be different from this version once it has been copyedited and typeset.
PLEASE CITE THIS ARTICLE AS DOI:10.1063/5.0006074

AIP Publishing

R. M. Richard, J. F. Gonthier, A. M. James, H. R. McAlexander, A. Kumar, M. Saitow, X. Wang, B. P. Pritchard, P. Verma, H. F. Schaefer, K. Patkowski, R. A. King, E. F. Valeev, F. A. Evangelista, J. M. Turney, T. D. Crawford, and C. D. Sherrill, J. Chem. Theory Comput. **13**, 3185 (2017).

[22] G. Kresse and J. Furthmüller, Phys. Rev. B **54**, 11169 (1996).

[23] G. Kresse and D. Joubert, Phys. Rev. B **59**, 1758 (1999).

[24] J. Enkovaara, C. Rostgaard, J. J. Mortensen, J. Chen, M. Dułak, L. Ferrighi, J. Gavnholt, C. Glinsvad, V. Haikola, H. A. Hansen, H. H. Kristoffersen, M. Kuisma, A. H. Larsen, L. Lehtovaara, M. Ljungberg, O. Lopez-Acevedo, P. G. Moses, J. Ojanen, T. Olsen, V. Petzold, N. A. Romero, J. Stausholm-Møller, M. Strange, G. A. Tritsaris, M. Vanin, M. Walter, B. Hammer, H. Häkkinen, G. K. H. Madsen, R. M. Nieminen, J. K. Nørskov, M. Puska, T. T. Rantala, J. Schiøtz, K. S. Thygesen, and K. W. Jacobsen, J. Phys. Condens. Matter **22**, 253202 (2010).

[25] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. Buongiorno Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. Dal Corso, S. de Gironcoli, P. Delugas, R. A. DiStasio, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E. Küçükbenli, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N. L. Nguyen, H.-V. Nguyen, A. Otero-de-la Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A. P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, and S. Baroni, J. Phys. Condens. Matter **29**, 465901 (2017).

[26] J. VandeVondele, M. Krack, F. Mohamed, M. Parrinello, T. Chassaing, and J. Hutter, Comput. Phys. Commun. **167**, 103 (2005).

[27] J. McClain, Q. Sun, G. K.-L. Chan, and T. C. Berkelbach, J. Chem. Theory Comput. **13**, 1209 (2017).

[28] Q. Sun, T. C. Berkelbach, J. D. McClain, and G. K.-L. Chan, J. Chem. Phys. **147**, 164119 (2017).

[29] S. Sharma and G. K.-L. Chan, J. Chem. Phys. **136**, 124121 (2012).

[30] S. Wouters, W. Poelmans, P. W. Ayers, and D. Van Neck, Comput. Phys. Commun. **185**, 1501 (2014).

[31] S. Sharma, A. A. Holmes, G. Jeanmairet, A. Alavi, and C. J. Umrigar, J. Chem. Theory Comput. **13**, 1595 (2017).

[32] J. E. T. Smith, B. Mussard, A. A. Holmes, and S. Sharma, J. Chem. Theory Comput. **13**, 5468 (2017).

[33] A. A. Holmes, N. M. Tubman, and C. J. Umrigar, J. Chem. Theory Comput. **12**, 3674 (2016).

[34] J. Li, M. Otten, A. A. Holmes, S. Sharma, and C. J. Umrigar, J. Chem. Phys. **149**, 214110 (2018).

[35] G. H. Booth, S. D. Smart, and A. Alavi, Mol. Phys. **112**, 1855 (2014).

[36] U. Bozkaya, J. M. Turney, Y. Yamaguchi, H. F. Schaefer, and C. D. Sherrill, J. Chem. Phys. **135**, 104103 (2011).

[37] R. Seeger and J. A. Pople, J. Chem. Phys. **66**, 3045 (1977).

[38] C. Van Wüllen, J. Comput. Chem. **23**, 779 (2002).

[39] S. Lehtola, C. Steigemann, M. J. Oliveira, and M. A. Marques, SoftwareX **7**, 1 (2018).

[40] U. Ekström, L. Visscher, R. Bast, A. J. Thorvaldsen, and K. Ruud, J. Chem. Theory Comput. **6**, 1971 (2010).

[41] O. A. Vydrov and T. Van Voorhis, J. Chem. Phys. **133**, 244103 (2010).

[42] Q. Sun, J. Comput. Chem. **36**, 1664 (2015).

[43] R. A. Friesner, Chem. Phys. Lett. **116**, 39 (1985).

[44] F. Neese, F. Wennmohs, A. Hansen, and U. Becker, Chem. Phys. **356**, 98 (2009).

[45] R. Izsák and F. Neese, J. Chem. Phys. **135**, 144105 (2011).

[46] S. Goedecker, M. Teter, and J. Hutter, Phys. Rev. B **54**, 1703 (1996).

[47] J. Hutter, M. Iannuzzi, F. Schiffmann, and J. VandeVondele, Wiley Interdiscip. Rev.: Comput. Mol. Sci. **4**, 15 (2014).

[48] C. Angeli, R. Cimiraglia, S. Evangelisti, T. Leininger, and J. P. Malrieu, J. Chem. Phys. **114**, 10252 (2001).

[49] S. Guo, M. A. Watson, W. Hu, Q. Sun, and G. K.-L. Chan, J. Chem. Theory Comput. **12**, 1583 (2016).

[50] S. R. Langhoff and E. R. Davidson, Int. J. Quantum Chem. **8**, 61 (1974).

[51] J. A. Pople, R. Seeger, and R. Krishnan, Int. J. Quantum Chem. **12**, 149 (1977).

[52] P. Knowles and N. Handy, Chem. Phys. Lett. **111**, 315 (1984).

[53] J. Olsen, P. Jørgensen, and J. Simons, Chem. Phys. Lett. **169**, 463 (1990).

[54] H. Sekino and R. J. Bartlett, Int. J. Quantum Chem. **26**, 255 (1984).

[55] A. C. Scheiner, G. E. Scuseria, J. E. Rice, T. J. Lee, and H. F. Schaefer, J. Chem. Phys. **87**, 5361 (1987).

[56] G. E. Scuseria, C. L. Janssen, and H. F. Schaefer, J. Chem. Phys. **89**, 7382 (1988).

[57] K. Raghavachari, G. W. Trucks, J. A. Pople, and M. Head-Gordon, Chem. Phys. Lett. **157**, 479 (1989).

[58] E. A. Salter, G. W. Trucks, and R. J. Bartlett, J. Chem. Phys. **90**, 1752 (1989).

[59] G. E. Scuseria, J. Chem. Phys. **94**, 442 (1991).

[60] M. Nooijen and R. J. Bartlett, J. Chem. Phys. **102**, 3629 (1995).

[61] H. Koch, A. S. De Merás, T. Helgaker, and O. Christiansen, J. Chem. Phys. **104**, 4157 (1996).

[62] M. Musial, S. A. Kucharski, and R. J. Bartlett, J. Chem. Phys. **118**, 1128 (2003).

[63] A. I. Krylov, Acc. Chem. Res. **39**, 83 (2006).

[64] H. J. Werner and P. J. Knowles, J. Chem. Phys. **82**, 5053 (1985).

[65] H. J. A. Jensen, P. Jørgensen, and H. Ågren, J. Chem. Phys. **87**, 451 (1987).

[66] J. Schirmer, Phys. Rev. A **26**, 2395 (1982).

[67] J. Schirmer, L. S. Cederbaum, and O. Walter, Phys. Rev. A **28**, 1237 (1983).

[68] J. Schirmer and A. B. Trofimov, J. Chem. Phys. **120**, 11449 (2004).

[69] A. Dreuw and M. Wormit, Wiley Interdiscip. Rev.: Comput. Mol. Sci. **5**, 82 (2015).

[70] S. Banerjee and A. Y. Sokolov, J. Chem. Phys. **151**, 224112 (2019).

[71] L. Hedin, Phys. Rev. **139**, A796 (1965).

[72] F. Aryasetiawan and O. Gunnarsson, Reports Prog. Phys. **61**, 237 (1998), arXiv:9712013 [cond-mat].

[73] X. Ren, P. Rinke, V. Blum, J. Wieferink, A. Tkatchenko, A. Sanfilippo, K. Reuter, and M. Scheffler, New J. Phys. **14** (2012), 10.1088/1367-2630/14/5/053020, arXiv:1201.0655.

[74] J. Wilhelm, M. Del Ben, and J. Hutter, J. Chem. Theory Comput. **12**, 3623 (2016).

[75] J. F. Stanton and J. Gauss, Theor. Chem. Acc. **93**, 303 (1996).

[76] J. C. Saeh and J. F. Stanton, J. Chem. Phys. **111**, 8275 (1999).

[77] M. Motta, D. M. Ceperley, G. K.-L. Chan, J. A. Gomez, E. Gull, S. Guo, C. A. Jiménez-Hoyos, T. N. Lan, J. Li, F. Ma, A. J. Millis, N. V. Prokof'ev, U. Ray, G. E. Scuseria, S. Sorella, E. M. Stoudenmire, Q. Sun, I. S. Tupitsyn, S. R. White, D. Zgid, and S. Zhang, Phys. Rev. X **7**, 031059 (2017).

[78] Q. Sun, J. Yang, and G. K.-L. Chan, Chem. Phys. Lett. **683**, 291 (2017).

[79] R. E. Thomas, Q. Sun, A. Alavi, and G. H. Booth, J. Chem. Theory Comput. **11**, 5316 (2015).

[80] X. Wang and T. C. Berkelbach, (2020), arXiv:2001.11050 [cond-mat.mtrl-sci], arXiv:2001.11050.

[81] Y. Gao, Q. Sun, J. M. Yu, M. Motta, J. McClain, A. F. White, A. J. Minnich, and G. K.-L. Chan, (2019), arXiv:1910.02191 [cond-mat.mtrl-sci], arXiv:1910.02191.

[82] S. H. Vosko, L. Wilk, and M. Nusair, Can. J. Phys. **58**, 1200 (1980).

[83] J. P. Perdew, K. Burke, and M. Ernzerhof, Phys. Rev. Lett. **77**, 3865 (1996), arXiv:0927-0256(96)00008 [10.1016].

[84] L. Visscher, T. Enevoldsen, T. Saue, H. J. A. Jensen, and J. Oddershede, J. Comput. Chem. **20**, 1262 (1999).

[85] T. Helgaker, M. Jaszuński, and K. Ruud, Chem. Rev. **99**, 293 (1999).

[86] T. Enevoldsen, L. Visscher, T. Saue, H. J. A. Jensen, and J. Oddershede, J. Chem. Phys. **112**, 3493 (2000).

[87] V. Sychrovský, J. Gräfenstein, and D. Cremer, J. Chem. Phys. **113**, 3530 (2000).

[88] T. Helgaker, M. Watson, and N. C. Handy, J. Chem. Phys. **113**, 9402 (2000).

[89] L. Cheng, Y. Xiao, and W. Liu, J. Chem. Phys. **130**, 144102 (2009).

[90] G. Schreckenbach and T. Ziegler, J. Phys. Chem. A **101**, 3388 (1997).

[91] F. Neese, J. Chem. Phys. **115**, 11080 (2001).

[92] Z. Rinkevicius, L. Telyatnyk, P. Salek, O. Vahtras, and H. Ågren, J. Chem. Phys. **119**, 10489 (2003).

[93] P. Hrobárik, M. Repiský, S. Komorovský, V. Hrobáriková, and M. Kaupp, Theor. Chem. Acc. **129**, 715 (2011).

[94] S. P. Sauer, J. Oddershede, and J. Geertsen, Mol. Phys. **76**, 445 (1992).

[95] J. Gauss, K. Ruud, and T. Helgaker, J. Chem. Phys. **105**, 2804 (1996).

[96] F. Neese, J. Chem. Phys. **118**, 3939 (2003).

The Journal of Chemical Physics ACCEPTED MANUSCRIPT

[97] A. V. Arbuznikov, J. Vaara, and M. Kaupp, J. Chem. Phys. **120**, 2127 (2004).

[98] R. Curl, Mol. Phys. **9**, 585 (1965).

[99] G. Tarczay, P. G. Szalay, and J. Gauss, J. Phys. Chem. A **114**, 9246 (2010).

[100] T. A. Keith, Chem. Phys. **213**, 123 (1996).

[101] R. Cammi, J. Chem. Phys. **109**, 3185 (1998).

[102] M. R. Pederson and S. N. Khanna, Phys. Rev. B **60**, 9566 (1999).

[103] F. Neese, J. Chem. Phys. **127**, 164112 (2007).

[104] S. Schmitt, P. Jost, and C. van Wüllen, J. Chem. Phys. **134**, 194113 (2011).

[105] F. London, J. Phys. le Radium **8**, 397 (1937).

[106] R. Ditchfield, Mol. Phys. **27**, 789 (1974).

[107] H. M. Petrilli, P. E. Blöchl, P. Blaha, and K. Schwarz, Phys. Rev. B **57**, 14690 (1998).

[108] S. Adiga, D. Aebi, and D. L. Bryce, Can. J. Chem. **85**, 496 (2007).

[109] J. Autschbach, S. Zheng, and R. W. Schurko, Concepts Magn. Reson. Part A **36A**, 84 (2010).

[110] K. G. Dyall, J. Chem. Phys. **106**, 9618 (1997).

[111] W. Kutzelnigg and W. Liu, J. Chem. Phys. **123**, 241102 (2005).

[112] W. Liu and D. Peng, J. Chem. Phys. **125**, 044102 (2006).

[113] M. Iliaš and T. Saue, J. Chem. Phys. **126**, 064102 (2007).

[114] L. Cheng and J. Gauss, J. Chem. Phys. **134**, 244112 (2011).

[115] G. Knizia, J. Chem. Theory Comput. **9**, 4834 (2013).

[116] A. E. Reed, R. B. Weinstock, and F. Weinhold, J. Chem. Phys. **83**, 735 (1985).

[117] Q. Sun and G. K.-L. Chan, J. Chem. Theory Comput. **10**, 3784 (2014).

[118] J. M. Foster and S. F. Boys, Rev. Mod. Phys. **32**, 300 (1960).

[119] C. Edmiston and K. Ruedenberg, J. Chem. Phys. **43**, S97 (1965).

[120] J. Pipek and P. G. Mezey, J. Chem. Phys. **90**, 4916 (1989).

[121] W. D. Derricotte and F. A. Evangelista, J. Chem. Theory Comput. **13**, 5984 (2017).

[122] Q. Sun, (2016), arXiv:1610.08423 [physics.chem-ph].

[123] G. Pizzi, V. Vitale, R. Arita, S. Blügel, F. Freimuth, G. Géranton, M. Gibertini, D. Gresch, C. Johnson, T. Koretsune, J. Ibañez-Azpiroz, H. Lee, J.-M. Lihm, D. Marchand, A. Marrazzo, Y. Mokrousov, J. I. Mustafa, Y. Nohara, Y. Nomura, L. Paulatto, S. Poncé, T. Ponweiser, J. Qiao, F. Thöle, S. S. Tsirkin, M. Wierzbowska, N. Marzari, D. Vanderbilt, I. Souza, A. A. Mostofi, and J. R. Yates, J. Phys. Condens. Matter **32**, 165902 (2020).

[124] A. Damle, L. Lin, and L. Ying, J. Chem. Theory Comput. **11**, 1463 (2015).

[125] A. Damle, L. Lin, and L. Ying, J. Comput. Phys. **334**, 1 (2017).

[126] A. Klamt and G. Schüürmann, J. Chem. Soc., Perkin Trans. 2 , 799 (1993).

[127] E. Cancès, B. Mennucci, and J. Tomasi, J. Chem. Phys. **107**, 3032 (1997).

[128] B. Mennucci, E. Cancès, and J. Tomasi, J. Phys. Chem. B **101**, 10506 (1997).

[129] E. Cancès, Y. Maday, and B. Stamm, J. Chem. Phys. **139**, 054111 (2013).

[130] F. Lipparini, B. Stamm, E. Cancès, Y. Maday, and B. Mennucci, J. Chem. Theory Comput. **9**, 3637 (2013).

[131] F. Lipparini, G. Scalmani, L. Lagardère, B. Stamm, E. Cancès, Y. Maday, J.-P. Piquemal, M. J. Frisch, and B. Mennucci, J. Chem. Phys. **141**, 184108 (2014).

[132] B. Stamm, E. Cancès, F. Lipparini, and Y. Maday, J. Chem. Phys. **144**, 054101 (2016).

[133] F. Lipparini and B. Mennucci, J. Chem. Phys. **144**, 160901 (2016).

[134] Z. Li, S. Guo, Q. Sun, and G. K.-L. Chan, Nat. Chem. **11**, 1026 (2019).

[135] M. Scheurer, P. Reinholdt, E. R. Kjellgren, J. M. Haugaard Olsen, A. Dreuw, and J. Kongsted, J. Chem. Theory Comput. **15**, 6154 (2019).

[136] M. Scheurer, "CPPE: C++ and Python Library for Polarizable Embedding," (2019), DOI: 10.5281/zenodo.3345696.

[137] W. Liu and D. Peng, J. Chem. Phys. **131**, 031104 (2009).

[138] R. Flores-Moreno, R. J. Alvarez-Mendez, A. Vela, and A. M. Köster, J. Comput. Chem. **27**, 1009 (2006).

[139] B. Mussard and S. Sharma, J. Chem. Theory Comput. **14**, 154 (2018).

[140] L.-P. Wang and C. Song, J. Chem. Phys. **144**, 214108 (2016).

[141] J. Hermann, "Pyberny," (2020), DOI: 10.5281/zenodo.3695038.

[142] S. Grimme, J. Antony, S. Ehrlich, and H. Krieg, J. Chem. Phys. **132**, 154104 (2010).

[143] J. L. C. Sainz, "A python wrapper for DFT-D3," https://github.com/cuanto/libdftd3 (Accessed 21 Feb 2020).

[144] D. A. Matthews, (2016), arXiv:1607.00291 [cs.MS], arXiv:1607.00291.

[145] J. Huang, D. A. Matthews, and R. A. van de Geijn, (2017), arXiv:1704.03092 [cs.MS], arXiv:1704.03092.

[146] D. Matthews, "TBLIS is a library and framework for performing tensor operations, especially tensor contraction, using efficient native algorithms," https://github.com/devinamatthews/tblis (Accessed 21 Feb 2020).

[147] J. Kim, A. D. Baczewski, T. D. Beaudet, A. Benali, M. C. Bennett, M. A. Berrill, N. S. Blunt, E. J. L. Borda, M. Casula, D. M. Ceperley, S. Chiesa, B. K. Clark, R. C. Clay, K. T. Delaney, M. Dewing, K. P. Esler, H. Hao, O. Heinonen, P. R. C. Kent, J. T. Krogel, I. Kylänpää, Y. W. Li, M. G. Lopez, Y. Luo, F. D. Malone, R. M. Martin, A. Mathuriya, J. McMinis, C. A. Melton, L. Mitas, M. A. Morales, E. Neuscamman, W. D. Parker, S. D. Pineda Flores, N. A. Romero, B. M. Rubenstein, J. A. R. Shea, H. Shin, L. Shulenburger, A. F. Tillack, J. P. Townsend, N. M. Tubman, B. Van Der Goetz, J. E. Vincent, D. C. Yang, Y. Yang, S. Zhang, and L. Zhao, J. Phys. Condens. Matter **30**, 195901 (2018).

[148] L. K. Wagner, K. Williams, S. Pathak, B. Busemeyer, J. N. B. Rodrigues, Y. Chang, A. Munoz, and C. Lorsung, "Python library for real space quantum Monte Carlo," https://github.com/WagnerGroup/pyqmc (Accessed 21 Feb 2020).

[149] L. K. Wagner, M. Bajdich, and L. Mitas, J. Comput. Phys. **228**, 3390 (2009).

[150] J. S. Spencer, N. S. Blunt, S. Choi, J. Etrych, M. A. Filip, W. M. Foulkes, R. S. Franklin, W. J. Handley, F. D. Malone, V. A. Neufeld, R. Di Remigio, T. W. Rogers, C. J. Scott, J. J. Shepherd, W. A. Vigor, J. Weston, R. Q. Xu, and A. J. Thom, J. Chem. Theory Comput. **15**, 1728 (2019).

[151] D. V. Chulhai and J. D. Goodpaster, J. Chem. Theory Comput. **13**, 1503 (2017).

[152] H. R. Petras, D. S. Graham, S. K. Ramadugu, J. D. Goodpaster, and J. J. Shepherd, J. Chem. Theory Comput. **15**, 5332 (2019).

[153] J. D. Goodpaster, D. S. Graham, and D. V. Chulhai, "Goodpaster/QSoME: Initial Release," (2019), DOI: 10.5281/zenodo.3356913.

[154] M. R. Hermes and L. Gagliardi, J. Chem. Theory Comput. **15**, 972 (2019).

[155] H. Q. Pham, M. R. Hermes, and L. Gagliardi, J. Chem. Theory Comput. **16**, 130 (2020).

[156] X. Zhang and E. A. Carter, J. Chem. Theory Comput. **15**, 949 (2019).

[157] Z.-H. Cui, T. Zhu, and G. K.-L. Chan, J. Chem. Theory Comput. **16**, 119 (2019).

[158] T. Zhu, Z. H. Cui, and G. K.-L. Chan, J. Chem. Theory Comput. **16**, 141 (2020).

[159] L. Gagliardi, D. G. Truhlar, G. Li Manni, R. K. Carlson, C. E. Hoyer, and J. L. Bao, Acc. Chem. Res. **50**, 66 (2017).

[160] J. J. Eriksen, "PyMBE: A Many-Body Expanded Correlation Code by Janus Juul Eriksen," Https://gitlab.com/januseriksen/pymbe (Accessed 21 Feb 2020).

[161] J. J. Eriksen, F. Lipparini, and J. Gauss, J. Phys. Chem. Lett. **8**, 4633 (2017), arXiv:1708.02103.

[162] J. J. Eriksen and J. Gauss, J. Chem. Theory Comput. **14**, 5180 (2018), arXiv:1807.01328.

[163] J. J. Eriksen and J. Gauss, J. Chem. Theory Comput. **15**, 4873 (2019), arXiv:1905.02786.

[164] J. J. Eriksen and J. Gauss, J. Phys. Chem. Lett. **10**, 7910 (2019), arXiv:1910.03527.

[165] S. Iskakov, A. A. Rusakov, D. Zgid, and E. Gull, Phys. Rev. B **100**, 085112 (2019).

[166] W. Li, C. Chen, D. Zhao, and S. Li, Int. J. Quantum Chem. **115**, 641 (2015).

[167] W. Li, Z. Ni, and S. Li, Mol. Phys. **114**, 1447 (2016).

[168] G. F. von Rudorff and O. A. von Lilienfeld, (2018), arXiv:1809.01647 [physics.chem-ph].

[169] G. F. Von Rudorff and O. A. Von Lilienfeld, J. Phys. Chem. B **123**, 10073 (2019).

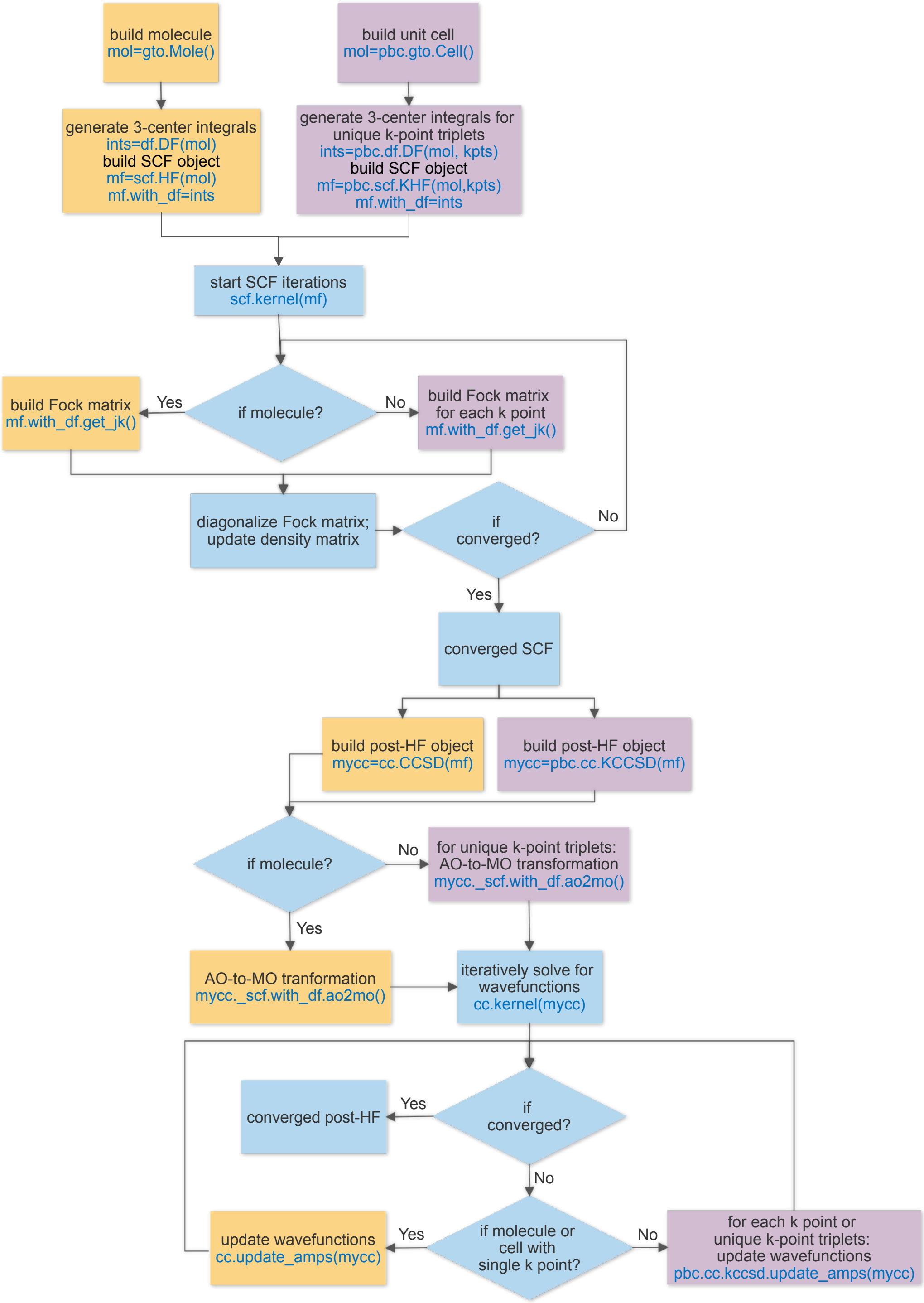
[170] P. Koval, M. Barbry, and D. Sánchez-Portal, Comput. Phys. Commun. **236**, 188 (2019).

[171] S. Schwalbe, L. Fiedler, T. Hahn, K. Trepte, J. Kraus, and J. Kortus, (2019), arXiv:1905.02631 [physics.comp-ph].

[172] M. F. Herbst, A. Dreuw, and J. E. Avery, J. Chem. Phys. **149**, 084106 (2018).

[173] Z. Rinkevicius, X. Li, O. Vahtras, K. Ahmadzadeh, M. Brand, M. Ringholm, N. H. List, M. Scheurer, M. Scott, A. Dreuw, and P. Norman, Wiley Interdiscip. Rev.: Comput. Mol. Sci. , e1457 (2019), DOI: 10.1002/wcms.1457.

[174] M. F. Herbst, M. Scheurer, T. Fransson, D. R. Rehn, and A. Dreuw, Wiley Interdiscip. Rev.: Comput. Mol. Sci. , e1462 (2020), DOI: 10.1002/wcms.1462.

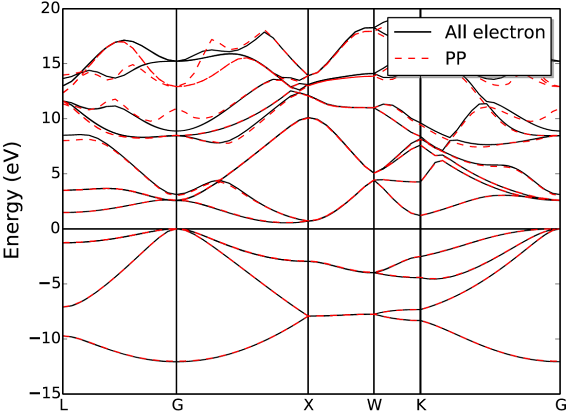

build molecule
mol=gto.Mole()
build unit cell
mol=pbc.gto.Cell()
generate 3-center integrals
ints=df.DF(mol)
build SCF object
mf=scf.HF(mol)
mf.with_df=ints
generate 3-center integrals for unique k-point triplets
ints=pbc.df.DF(mol, kpts)
build SCF object
mf=pbc.scf.KHF(mol,kpts)
mf.with_df=ints
start SCF iterations
scf.kernel(mf)
build Fock matrix
mf.with_df.get_jk()
Yes
if molecule?
No
build Fock matrix for each k point
mf.with_df.get_jk()
diagonalize Fock matrix; update density matrix
if converged?
No
Yes
converged SCF
build post-HF object
mycc=cc.CCSD(mf)
build post-HF object
mycc=pbc.cc.KCCSD(mf)
if molecule?
No
for unique k-point triplets: AO-to-MO transformation
mycc._scf.with_df.ao2mo()
Yes
AO-to-MO tranformation
mycc._scf.with_df.ao2mo()
iteratively solve for wavefunctions
cc.kernel(mycc)
converged post-HF
Yes
if converged?
No
update wavefunctions
cc.update_amps(mycc)
Yes
if molecule or cell with single k point?
No
for each k point or unique k-point triplets: update wavefunctions
pbc.cc.kccsd.update_amps(mycc)

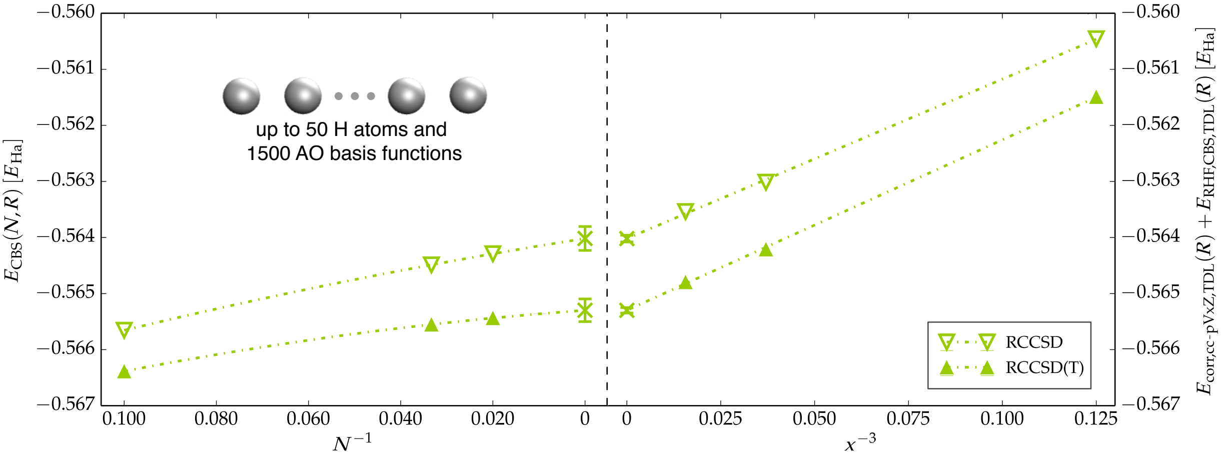
```
import numpy
import pyscf
mol = pyscf.M()
n = 10
mol.nelectron = n

# Define model Hamiltonian: tight binding on a ring
h1 = numpy.zeros((n, n))
for i in range(n-1):
    h1[i, i+1] = h1[i+1, i] = -1.
h1[n-1, 0] = h1[0, n-1] = -1.

# Build the 2-electron interaction tensor starting from a random 3-index tensor.
tensor = numpy.random.rand(2, n, n)
tensor = tensor + tensor.transpose(0, 2, 1)
eri = numpy.einsum('xpq,xrs->pqrs', tensor, tensor)

# SCF for the custom Hamiltonian
mf = mol.HF()
mf.get_hcore = lambda *args: h1
mf.get_ovlp = lambda *args: numpy.eye(n)

# Option 1: overwrite the attribute mf._eri for the 2-electron interactions
mf._eri = eri
mf.run()

# Option 2: introduce the 2-electron interaction through the Cholesky decomposed tensor.
dfmf = mf.density_fit()
dfmf.with_df._cderi = tensor
dfmf.run()

# Option 3: define a custom HF potential method
def get_veff(mol, dm):
    J = numpy.einsum('xpq,xrs,pq->rs', tensor, tensor, dm)
    K = numpy.einsum('xpq,xrs,qr->ps', tensor, tensor, dm)
    return J - K * .5
mf.get_veff = get_veff
mf.run()

# Call the second order SCF solver in case converging the DIIS-driven HF method
# without a proper initial guess is difficult.
mf = mf.newton().run()

# Run post-HF methods based on the custom SCF object
mf.MP2().run()
mf.CISD().run()
mf.CCSD().run()
mf.CASSCF(4, 4).run()
mf.CASCI(4, 4).run().NEVPT2().run()
mf.TDHF().run()
mf.CCSD().run().EOMIP().run()
mc = shci.SHCISCF(mf, 4, 4).run()
mc = dmrgscf.DMRGSCF(mf, 4, 4).run()
```

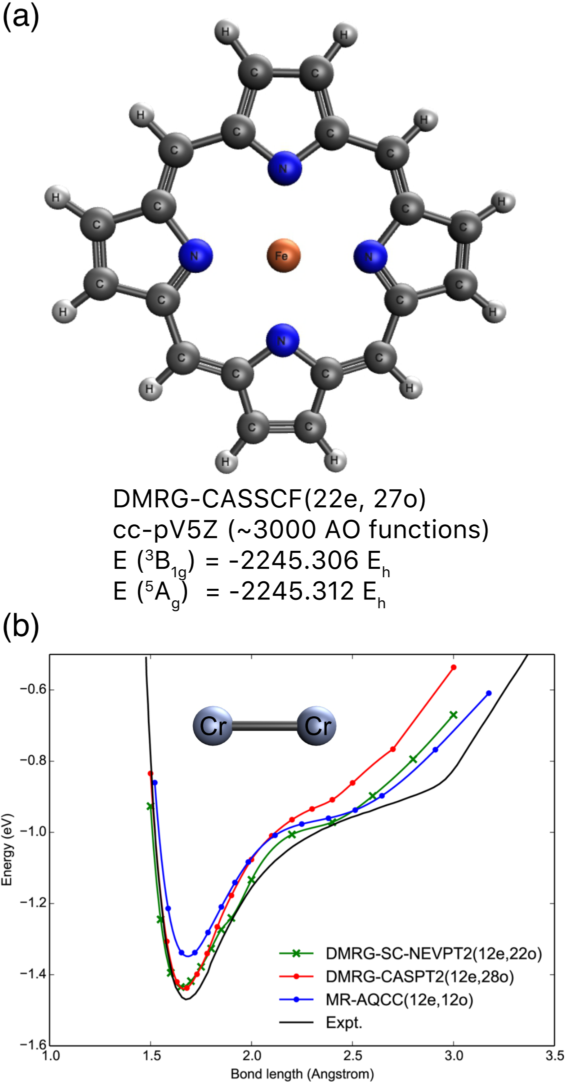
```
import numpy
import pyscf
class MP2AsFCISolver(object):
    def kernel(self, h1, h2, norb, nelec, ci0=None, ecore=0, **kwargs):
        # Kernel takes the set of integrals from the current set of orbitals
        fakemol = pyscf.M(verbose=0)
        fakemol.nelectron = sum(nelec)
        fake_hf = fakemol.RHF()
        fake_hf._eri = h2
        fake_hf.get_hcore = lambda *args: h1
        fake_hf.get_ovlp = lambda *args: numpy.eye(norb)

        # Build an SCF object fake_hf without SCF iterations to perform MP2
        fake_hf.mo_coeff = numpy.eye(norb)
        fake_hf.mo_occ = numpy.zeros(norb)
        fake_hf.mo_occ[:fakemol.nelectron//2] = 2
        self.mp2 = fake_hf.MP2().run()
        return self.mp2.e_tot + ecore, self.mp2.t2

    def make_rdm12(self, t2, norb, nelec):
        dm1 = self.mp2.make_rdm1(t2)
        dm2 = self.mp2.make_rdm2(t2)
        return dm1, dm2

mol = pyscf.M(atom='H 0 0 0; F 0 0 1.1', basis='ccpvdz')
mf = mol.RHF().run()
# Put in the active space all orbitals of the system
mc = pyscf.mcscf.CASSCF(mf, mol.nao, mol.nelectron)
mc.fcisolver = MP2AsFCISolver()
# Internal rotation inside the active space needs to be enabled
mc.internal_rotation = True
mc.kernel()
```

```
import pyscf
frag = pyscf.M(atom=’frag.xyz’, basis=’ccpvtz’)
env  = pyscf.M(atom=’env.xyz’, basis=’sto-3g’)
sys = frag + env

def embedding_gradients(sys):
    # Regular HF energy and nuclear gradients of the entire system
    sys_hf = sys.HF().run()
    grad_sys = sys_hf.Gradients().kernel()

    # Construct a CASCI-like effective 1-electron Hamiltonian for the fragment
    # with the presence of outlying atoms in the environment. dm_env is the
    # density matrix in the environment block
    dm_env = sys_hf.make_rdm1()
    dm_env[frag.nao:,:] = dm_env[:,frag.nao:] = 0
    frag_hcore_eff = (sys_hf.get_hcore() + sys_hf.get_veff(sys, dm_env))[:frag.nao, :frag.nao]

    # Customize the zeroth order calculation by overwriting the core Hamiltonian.
    # HF and CISD now provide the embedding wavefunction on fragment.
    geom_frag = sys.atom_coords(unit=’Angstrom’)[:frag.natm]
    frag.set_geom_(geom_frag)
    frag_hf = frag.HF()
    frag_hf.get_hcore = lambda *args: frag_hcore_eff
    frag_hf.run()()
    frag_ci = frag_hf.CISD().run()

    # The .Gradients() method enables a regular analytical nuclear gradient object
    # to evaluate the Hellmann-Feynman forces on fragment using the first order
    # derivatives of the original fragment Hamiltonian and the variational
    # embedding wavefunction.
    grad_hf_frag = frag_hf.Gradients().kernel()
    grad_ci_frag = frag_ci.Gradients().kernel()

    # Approximate the energy and gradients of the entire system with the post-HF
    # correction on fragment
    approx_e = sys_hf.e_tot + frag_ci.e_tot - frag_hf.e_tot
    approx_grad = grad_sys
    approx_grad[:frag.natm] += grad_ci - grad_hf
    print(’Approximate gradients:\n’, approx_grad)
    return approx_e, approx_grad

new_sys = pyscf.geomopt.as_pyscf_method(sys,\
          embedding_gradients).Gradients().optimizer().kernel()
```

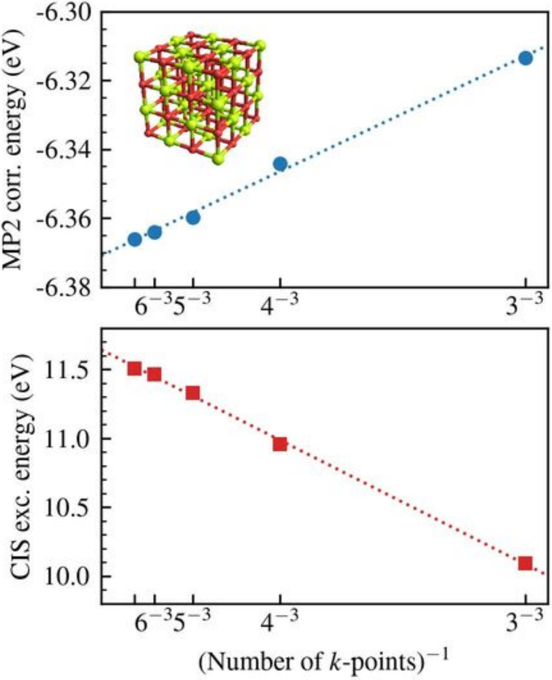
```
import pyscf
mol = pyscf.M(atom = ’N 0 0 0; N 0 0 1.1’ , basis = ’ccpvdz’)
mf = mol.RKS()
mf.xc =’CAMB3LYP’
mf.xc = ’’’0.19*SR_HF(0.33) + 0.65*LR_HF(0.33) + 0.46*ITYH + 0.35*B88, 0.19*VWN5 + 0.81*LYP’’’
mf.xc = ’RSH(0.33, 0.65, -0.46) + 0.46*ITYH + 0.35*B88, 0.19*VWN5 + 0.81*LYP’
e_mf = mf.kernel()
```

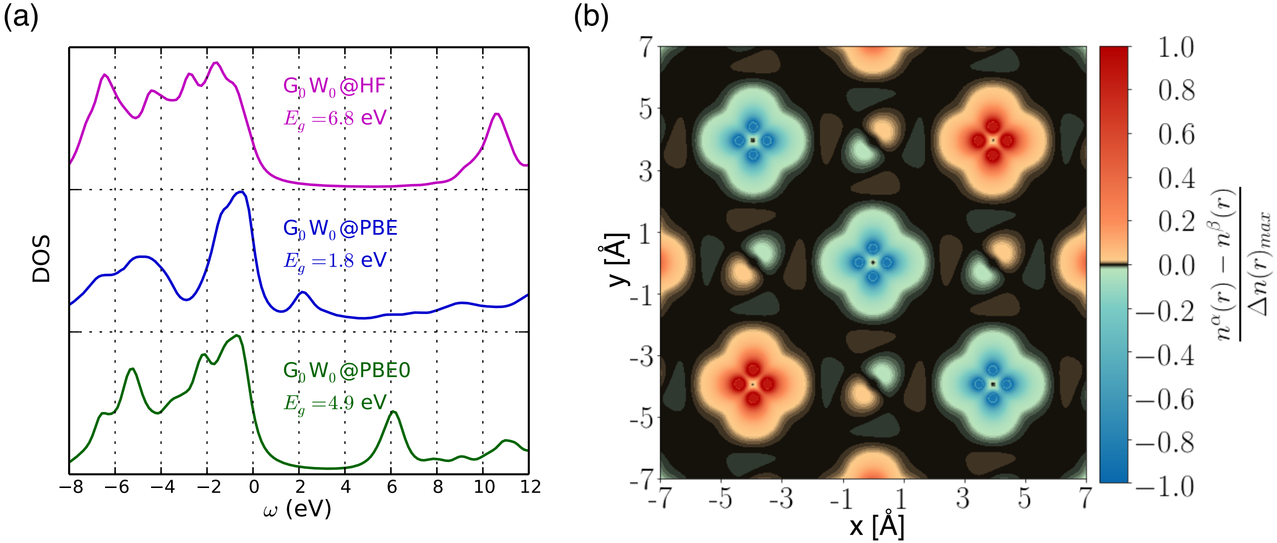

Energy (eV)
20
15
10
5
0
−5
−10
−15
L
G
X
W
K
G
All electron
PP

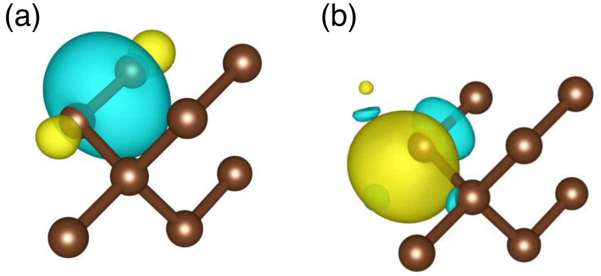

−0.560
−0.561
−0.562
−0.563
−0.564
−0.565
−0.566
−0.567
$E_{\mathrm{CBS}}(N,R)\ [E_{\mathrm{Ha}}]$
up to 50 H atoms and
1500 AO basis functions
0.100
0.080
0.060
0.040
0.020
0
N^{-1}
0
0.025
0.050
0.075
0.100
0.125
x^{-3}
RCCSD
RCCSD(T)
$E_{\mathrm{corr,cc\text{-}pVxZ,TDL}}(R) + E_{\mathrm{RHF,CBS,TDL}}(R)\ [E_{\mathrm{Ha}}]$

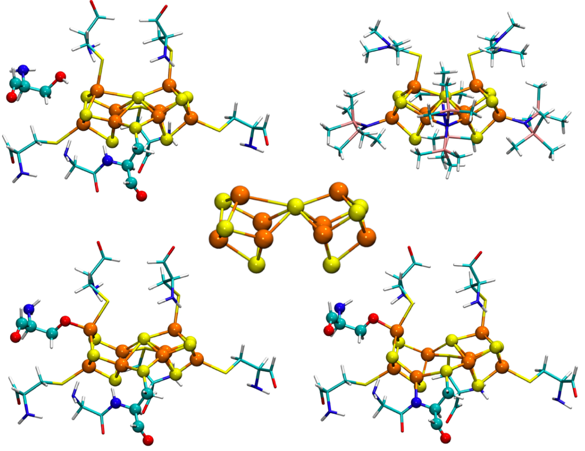

(a)
H
C
N
Fe
DMRG-CASSCF(22e, 27o)
cc-pV5Z (~3000 AO functions)
E ($^3B_{1g}$) = -2245.306 E_h
E (^5A_g) = -2245.312 E_h
(b)
Cr
Cr
Energy (eV)
-0.6
-0.8
-1.0
-1.2
-1.4
-1.6
1.0
1.5
2.0
2.5
3.0
3.5
Bond length (Angstrom)
DMRG-SC-NEVPT2(12e,22o)
DMRG-CASPT2(12e,28o)
MR-AQCC(12e,12o)
Expt.

```
import pyscf
cell = pyscf.M(atom=..., a=...) # ’a’ defines lattice vectors
mf = cell.HF(kpt = [0.23,0.23,0.23]).run()
# Use PBC CCSD class so integrals are handled
# correctly with respect to Coulomb divergences
mycc = pyscf.pbc.cc.CCSD(mf)
# molecular CCSD code used to compute correlation energy at single k-point
converged, ecorr = pyscf.cc.ccsd.kernel(mycc)
```

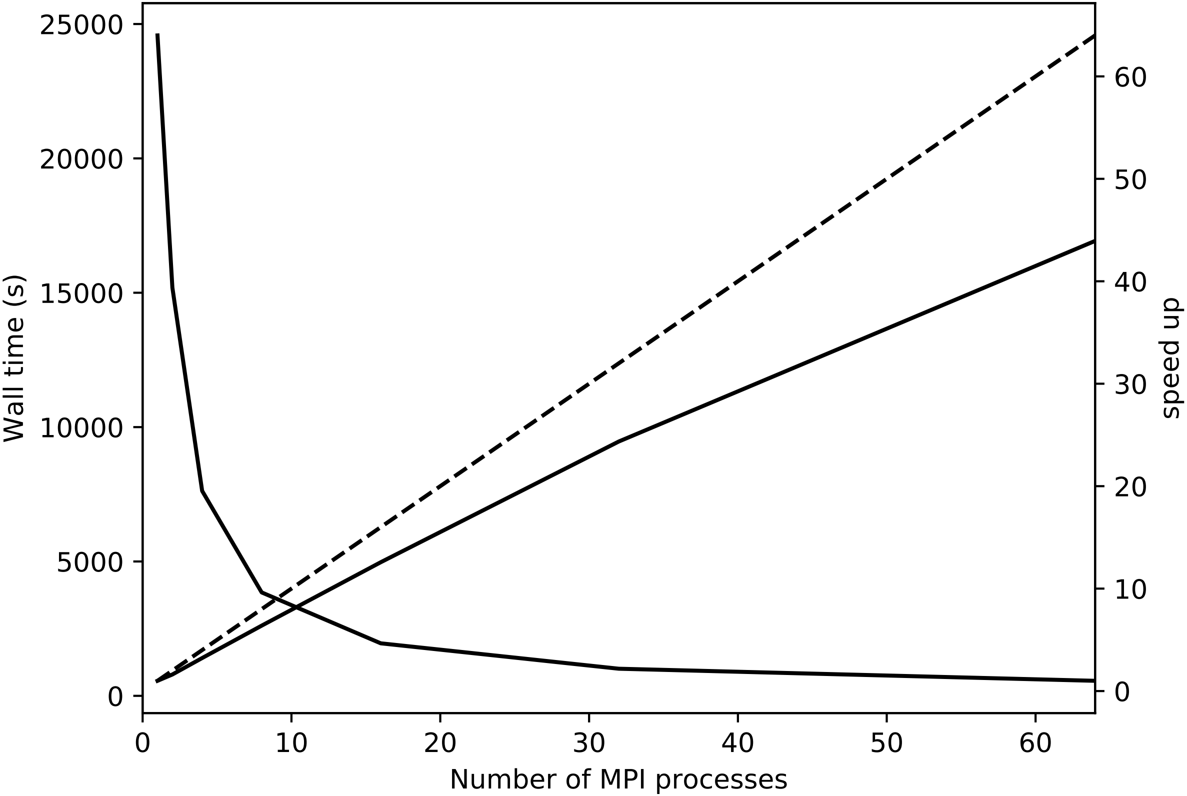

MP2 corr. energy (eV)
-6.30
-6.32
-6.34
-6.36
-6.38
6^{-3}
5^{-3}
4^{-3}
3^{-3}
CIS exc. energy (eV)
11.5
11.0
10.5
10.0
6^{-3}
5^{-3}
4^{-3}
3^{-3}
(Number of k-points)$^{-1}$


(a)
DOS
G0W0@HF
Eg = 6.8 eV
G0W0@PBE
Eg = 1.8 eV
G0W0@PBE0
Eg = 4.9 eV
−8
−6
−4
−2
0
2
4
6
8
10
12
ω (eV)
(b)
y [Å]
7
5
3
1
-1
-3
-5
-7
x [Å]
-7
-5
-3
-1
1
3
5
7
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
(n^α(r) − n^β(r)) / Δn(r)_max


(a)
(b)



```
# run in cmdline:
# mpirun -np 4 python input.py

import pyscf
mol = pyscf.M(...)

# Serial task
from pyscf import dft
mf = dft.RKS(mol).run(xc=’b3lyp’)
J, K = mf.get_jk(mol, mf.make_rdm1())

# MPI-parallel task
from mpi4pyscf import dft
mf = dft.RKS(mol).run(xc=’b3lyp’)
J, K = mf.get_jk(mol, mf.make_rdm1())
```


Wall time (s)
25000
20000
15000
10000
5000
0
speed up
60
50
40
30
20
10
0
0
10
20
30
40
50
60
Number of MPI processes