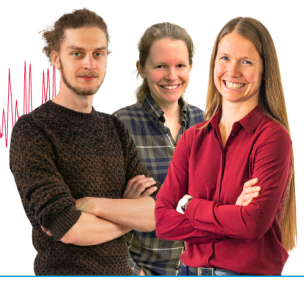
RESEARCH ARTICLE | NOVEMBER 22 2022

Differentiable quantum chemistry with PySCF for molecules and materials at the mean-field level and beyond

Xing Zhang ; Garnet Kin-Lic Chan

J. Chem. Phys. 157, 204801 (2022)
https://doi.org/10.1063/5.0118200
CHORUS



The Journal of Chemical Physics ARTICLE scitation.org/journal/jcp

Differentiable quantum chemistry with PySCF for molecules and materials at the mean-field level and beyond

Cite as: J. Chem. Phys. 157, 204801 (2022); doi: 10.1063/5.0118200
Submitted: 4 August 2022 • Accepted: 31 October 2022 •
Published Online: 22 November 2022

Xing Zhang and Garnet Kin-Lic Chan[a)]

AFFILIATIONS

Division of Chemistry and Chemical Engineering, California Institute of Technology, Pasadena, California 91125, USA

[a)]Author to whom correspondence should be addressed: gkc1000@gmail.com

ABSTRACT

We introduce an extension to the PySCF package, which makes it automatically differentiable. The implementation strategy is discussed, and example applications are presented to demonstrate the automatic differentiation framework for quantum chemistry methodology development. These include orbital optimization, properties, excited-state energies, and derivative couplings, at the mean-field level and beyond, in both molecules and solids. We also discuss some current limitations and directions for future work.

I. INTRODUCTION

Automatic differentiation (AD)[1,2] has recently become ubiquitous through its adoption in machine learning applications. AD removes the burden of implementing derivatives of complex computations by applying the chain rule to the elementary operations and functions of a computational graph. This can be used to automatically compute the exact derivatives of an arbitrary order.

In quantum chemistry, derivative evaluations appear in many types of calculations, such as in wavefunction optimization, to compute response properties and to obtain critical points and reaction paths on potential energy surfaces, in molecular dynamics simulations, for basis set and pseudopotential optimization, etc. Before the advent of AD, these derivatives were mainly computed by analytic schemes, where the symbolic formula for the derivative is first required (and which can be tedious to obtain), or computed numerically using finite difference methods (which are computationally expensive and prone to numerical instabilities). Both sets of difficulties are circumvented by applying AD, as shown in a few very recent works. For example, Tamayo-Mendoza *et al.*[3] implemented a fully differentiable Hartree–Fock (HF) method with AD (although the implementation was limited to minimal basis sets); Song *et al.*[4] introduced an AD scheme to compute nuclear gradients for tensor hyper-contraction based methods; Abbott *et al.*[5] applied AD to calculations of higher order nuclear derivatives with methods such as HF, second-order Møller–Plesset perturbation theory (MP2), and coupled cluster theory with single, double, and perturbative triple excitations [CCSD(T)]; and Kasim *et al.*[6] developed a differentiable quantum chemistry code called DQC for basis set optimization, molecular property calculations, etc., at the mean-field level. In addition, there are other quantum chemistry-related applications where AD has played an important role, e.g., in training neural network-based density functionals,[7,8] in differentiable tensor network algorithms,[9] and in quantum circuit optimization.[10]

A central mission of PySCF[11,12] is to provide a development platform to accelerate the implementation of new methods by its users. To further its potential, we have attempted to incorporate into PySCF the full functionality of AD. The resulting AD framework, which we call PySCFAD in the following, is available as an add-on to PySCF.[13] Although still in active development, we have found that PySCFAD is already very useful for derivative calculations associated with complex computational workflows. The purpose of this paper is thus to describe the current implementation of AD within PySCFAD and to illustrate its use in a range of applications.

The remainder of the paper is organized as follows: In Sec. II, we discuss the implementation of a few key components in PySCFAD. In Sec. III, example applications are presented to illustrate the capabilities of PySCFAD. In Sec. IV, we examine the computational efficiency of PySCFAD. In Sec. V, we summarize our

experience with AD in this project and outline some directions for future work.

II. IMPLEMENTATION

We now describe the implementation of PySCFAD. We will assume some knowledge of the basics of AD; for further introduction, see Ref. 2. In PySCF, the majority of methods are implemented using NumPy[14] and SciPy[15] functions. These functions have differentiable counterparts provided by several advanced AD libraries.[16–18] (Here, differentiable means that the function (or data) can be used or traced by an AD library in the computation of derivatives.) Thus, in many scenarios, transforming PySCF into PySCFAD is simply a matter of replacing the NumPy and SciPy functions with the corresponding differentiable ones from the AD library. However, there are also computationally heavy components such as the electron repulsion integrals (ERIs), which are implemented as C code in PySCF. We realize AD for these parts by registering the C functions that implement their analytic derivatives, which allows them to be called during the AD traversal of the computational graph. This not only avoids duplicate implementations of the same functionality but also reduces the cost of AD tracing through complex numerical algorithms. Note, however, that such a strategy only allows for derivatives of finite order, as the C derivative functions appear as black boxes to the AD framework.

Currently, we use Jax[18] as the backend AD package for PySCFAD. This is because, besides being an AD library, Jax also provides other appealing features, such as vectorization, parallelization, and just-in-time compilation, including for hardware accelerators. Using the strategy above, the following components of PySCF have been transformed to be compatible with automatic differentiation:

- molecular and crystal structures, Gaussian orbital evaluations, and ERIs (*gto* and *pbc*/*gto*),
- molecular and plane wave density fitting routines (*df* and *pbc*/*df*),
- HF and density functional theory (DFT) for molecules and solids (*scf*, *dft*, *pbc*/*scf*, and *pbc*/*dft*),
- time-dependent HF and DFT (*tdscf*),
- MP2 (*mp*),
- random-phase approximation (*gw*),
- coupled cluster theory (*cc*), and
- full configuration interaction (*fci*).

(The relevant modules in PySCF are listed in parentheses.) In the following, we give more details about the implementation of some of them.

A. Electron repulsion integrals

PySCF uses contracted Gaussian basis functions, the radial parts of which can be expressed as

$$\phi(\mathbf{r}) = \sum_i C_i (\mathbf{r}-\mathbf{r}_0)^l \exp(-\alpha_i (\mathbf{r}-\mathbf{r}_0)^2), \tag{1}$$

where l is the angular momentum and the three parameters $\mathbf{r}_0$, α_i, and C_i label the center, exponents, and contraction coefficients, respectively. Differentiation of a basis function with respect to these variables leads to another Gaussian function. As such, derivatives of an ERI can be computed by a sequence of similar integral evaluations.

In the current implementation of PySCFAD, the program walks through the computation graph and determines the required intermediate ERIs. For example, computing the second-order derivative of the one-electron overlap integral with respect to the basis function centers requires the evaluation of three integrals,

$$\nabla_{\mathbf{r}_0} \cdot \nabla_{\mathbf{r}_0} (\phi_\mu|\phi_\nu) \to (\nabla_{\mathbf{r}_0}\phi_\mu|\phi_\nu) \to \begin{cases} (\nabla_{\mathbf{r}_0}\phi_\mu|\nabla_{\mathbf{r}_0}\phi_\nu) \\ (\nabla^2_{\mathbf{r}_0}\phi_\mu|\phi_\nu), \end{cases}$$

where in the above, the permutation symmetries have been considered. In the implementation, these integrals are all evaluated analytically by the highly optimized integral library Libcint.[19] Thus, the ERIs themselves are treated as elementary functions during the AD procedure instead of the lower-level arithmetic operations that compute them.

For the commonly used ERIs, up to fourth-order derivatives with respect to the basis function centers are available through Libcint. If higher-order derivatives are needed, one can extend Libcint by generating the code to compute the required intermediate integrals before runtime with the accompanying automatic code generator. Only first-order derivatives with respect to the exponents and contraction coefficients have been implemented so far. This should be sufficient for many purposes, such as basis function optimization.

The procedure above, in a strict sense, is not fully differentiable because the ERI evaluation is a black box. However, a general, efficient, and fully differentiable implementation for the ERIs remains challenging. The advent of compiler-level AD tools[20] may facilitate such developments in the future.

B. Density functionals

A semi-local exchange–correlation (XC) density functional can be expressed as

$$E_{\mathrm{xc}} = \int d\mathbf{r} \rho(\mathbf{r}) \varepsilon_{\mathrm{xc}}[\rho(\mathbf{r}), \nabla\rho(\mathbf{r}), \nabla^2\rho(\mathbf{r}), \tau(\mathbf{r})], \tag{2}$$

where $\varepsilon_{\mathrm{xc}}$ is the XC energy per particle and ρ and τ label the electron density and the non-interacting kinetic energy density, respectively. The integration in Eq. (2) is usually carried out numerically over a quadrature grid due to the complexity of XC functionals,

$$E_{\mathrm{xc}} = \sum_i w_i \rho(\mathbf{r}_i) \varepsilon_{\mathrm{xc}}(\mathbf{r}_i), \tag{3}$$

where w_i is the weight for the ith grid point at position $\mathbf{r}_i$. The AD of E_{xc} is straightforward if one uses a fully differentiable implementation of $\varepsilon_{\mathrm{xc}}$. However, we do not pursue that strategy here, since the density derivatives to finite order (usually up to fourth order) are sufficient in most practical scenarios, and these have already been implemented in efficient density functional libraries such as Libxc.[21] Therefore, we take an approach similar to that introduced above for the AD of ERIs, where the derivatives of $\varepsilon_{\mathrm{xc}}$ with respect to

the density variables are computed analytically while other derivatives (e.g., the derivatives of the density variables) are generated by AD. In particular, LIBXC provides analytic functional derivatives of E_{xc}, from which the derivatives of ε_{xc} can be easily expressed. For example, the first-order derivative of ε_{xc} with respect to ρ within the local density approximation (LDA) is obtained as

$$\frac{\partial \varepsilon_{xc}}{\partial \rho} = \frac{v_{xc} - \varepsilon_{xc}}{\rho}, \tag{4}$$

where both ε_{xc} and v_{xc} (the XC potential) are computed analytically by LIBXC. The higher-order derivatives are obtained by recursion, e.g.,

$$\frac{\partial^2 \varepsilon_{xc}}{\partial \rho^2} = \frac{1}{\rho}\left(f_{xc} - 2\frac{\partial \varepsilon_{xc}}{\partial \rho}\right), \tag{5}$$

where f_{xc} is the XC kernel.

C. Eigendecompositions

Although JAX provides a differentiable implementation of the eigendecomposition, it does not handle degenerate eigenstates. Usually, in order to determine the derivatives of degenerate eigenstates, one needs to diagonalize the perturbations of the matrix being decomposed within the degenerate subspace until the degeneracy is removed. Such an approach, however, is not well defined for reverse-mode AD because the basis in which to expand the degenerate eigenstates is undetermined until the back propagation is carried out. The development of a general differentiable eigensolver is beyond the scope of the current work. In the following, we only discuss a workaround for the AD of eigenvalue problems arising in mean-field calculations.

At the mean-field level, a gauge invariant quantity will only respond to perturbations that mix occupied and unoccupied states. Thus, in many cases, the response of degenerate single-particle eigenstates, i.e., orbitals (which can only be either occupied or unoccupied for gapped systems), does not contribute to the derivatives and can be ignored. Our implementation directly follows the standard analytic formalism derived from linear response theory.[22] For a generalized eigenvalue problem

$$\mathbf{FC} = \mathbf{SC}\boldsymbol{\varepsilon}, \tag{6}$$

where $\boldsymbol{\varepsilon}$ and $\mathbf{C}$ denote the eigenvalue and eigenvector matrices, respectively, their differentials can be expressed as

$$\partial \varepsilon_{ii} = \left[\mathbf{I} \circ (\mathbf{C}^\dagger \partial \mathbf{F} \mathbf{C} - \mathbf{C}^\dagger \partial \mathbf{S} \mathbf{C} \boldsymbol{\varepsilon})\right]_{ii} \tag{7}$$

and

$$\partial \mathbf{C} = \mathbf{C}\left[\mathbf{W} \circ (\mathbf{C}^\dagger \partial \mathbf{F} \mathbf{C} - \mathbf{C}^\dagger \partial \mathbf{S} \mathbf{C} \boldsymbol{\varepsilon}) - \mathbf{I} \circ \left(\frac{1}{2}\mathbf{C}^\dagger \partial \mathbf{S} \mathbf{C}\right)\right], \tag{8}$$

respectively. In the two equations above, $\circ$ represents element-wise multiplication,

$$I_{ij} = \begin{cases} 1 & \text{if } i = j \text{ or } \varepsilon_i = \varepsilon_j, \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

and

$$W_{ij} = \begin{cases} 0 & \text{if } i = j \text{ or } \varepsilon_i = \varepsilon_j, \\ \dfrac{1}{\varepsilon_j - \varepsilon_i} & \text{otherwise.} \end{cases} \tag{10}$$

Similar approaches have also been reported in other works.[8,9]

D. Implicit differentiation of iterative solvers

Automatic differentiation of optimization problems or iterative solvers is usually done in one of two ways, namely, unrolling the iterations[1,23] or implicit differentiation.[2,24,25]

In the first approach, the entire set of iterations is differentiated, leading to a memory complexity that scales linearly with the number of iterations for reverse-mode AD. In addition, the so-obtained derivatives are usually initial guess-dependent. This can be understood with the example of computing the molecular orbital (MO) response [Eq. (8)]. Suppose in an extreme case that the self-consistent field (SCF) iteration takes the converged MOs as the initial guess, then the SCF will converge after one Fock matrix diagonalization. Due to the fact the SCF iteration is short-circuited (the Fock matrix is not rebuilt), the input MOs have no knowledge of their own derivatives, and the Fock matrix, which responds to the MO changes, will have the wrong derivative after this single SCF iteration, which in turn leads to the wrong MO response. In other words, self-consistency is not reached when solving Eq. (8), where $\partial\mathbf{F}$ depends on $\partial\mathbf{C}$; the quality of convergence of the self-consistent response equations is controlled only by the threshold of the SCF itself. Such errors can be corrected by increasing the number of SCF iterations so that the Fock matrix response with respect to the MO changes is gradually restored and the self-consistency of Eq. (8) is achieved. To see this effect, we computed the nuclear gradient of the N_2 molecule at the MP2 level, where the derivatives of the MO coefficients are evaluated by the scheme of unrolling the SCF iterations. Using the same set of converged MOs as the initial guess, we vary the number of SCF iterations and plot the gradient error, shown as the red curve in Fig. 1. It is clear that the computed gradient is inaccurate unless a sufficient number of SCF iterations are carried out to recover the correct MO response. Nevertheless, the method is problematic when performing AD for optimizations or for iterative solvers, as one has no direct control over the convergence of the computed derivatives.

In implicit differentiation, instead of differentiating the solver iterations, the optimality condition is implicitly differentiated. For example, the solution (denoted as $x^\star$) of the iterative solver should also be the root of some optimality condition,

$$f(x^\star(\theta), \theta) = 0. \tag{11}$$

According to the implicit function theorem,[2,26] $x^\star$ can be seen as an implicit function of θ, and its derivative can be obtained by solving a set of linear equations

$$\frac{\partial f}{\partial x^\star}\frac{\partial x^\star}{\partial \theta} = -\frac{\partial f}{\partial \theta}. \tag{12}$$

In the problem of computing the MO response, Eq. (12) simply corresponds to the coupled perturbed Hartree–Fock (CPHF)

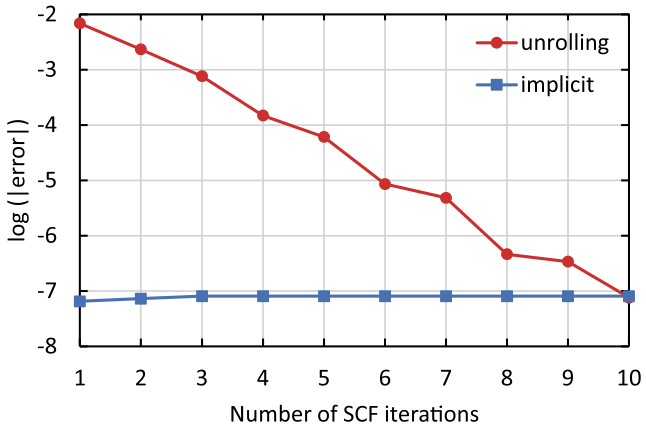


FIG. 1. The logarithm of the energy nuclear gradient errors plotted as a function of the number of SCF iterations for the N_2 molecule at the MP2/cc-pVTZ level. The SCF calculations use the converged MOs as the initial guess, and two AD approaches were carried out, namely, unrolling the iterations and implicit differentiation, whose results are shown as the red and blue curves, respectively.

equations.[22] More interestingly, in the reverse-mode AD, one does not directly solve Eq. (12); instead, the vector Jacobian product (i.e., $v^\top J$, where v is a vector and $J \equiv \frac{\partial x^\star}{\partial \theta}$) is computed. If we define $A \equiv \frac{\partial f}{\partial x^\star}$ and $B \equiv -\frac{\partial f}{\partial \theta}$, then the vector Jacobian product is obtained as

$$v^\top J = z^\top B, \tag{13}$$

where

$$A^\top z = v. \tag{14}$$

In this case, only one set of linear equations [Eq. (14)] needs to be solved, even if derivatives are evaluated with respect to multiple variables (i.e., when B changes but not A or v). Note that Eqs. (13) and (14) correspond exactly to the Z-vector approach[27] in conventional analytic derivative methods.

The advantages of the implicit differentiation approach are obvious. First, because only the optimality condition is differentiated, the computational complexity and memory footprint do not depend on the actual implementation of the solver or on the number of solver iterations. A notable benefit of this is that algorithms to accelerate convergence can be readily applied, such as the direct inversion of the iterative subspace[28] (DIIS) method, without any modification to the AD implementation. [Note, however, that Eq. (14) itself needs to be solved iteratively, which can be more expensive than the approach of unrolling the iterations, depending on the size and convergence of the problem.] Second, the computed derivatives have errors that are governed only by the accuracy of the solution of Eq. (14), which can be controlled with a predefined convergence threshold. This can be seen from the blue curve in Fig. 1, where the implicit differentiation approach is applied for the same MP2 nuclear gradient calculation as discussed above.

Given a general implementation,[25] implicit differentiations can be applied to almost any iterative solver. Currently, we have adapted it for the AD of SCF iterations and the coupled cluster (CC) amplitude equations. This eliminates the need to explicitly implement and solve the CPHF equations and the CC Λ equations.[29] Finally, it is interesting to mention that if the optimality condition being differentiated involves the fixed point problem of gauge variant quantities (e.g., wavefunctions), it is important to fix the gauge. In particular, the fixed point of the optimality condition should be identical to the variable used to evaluate the objective function. For example, when computing the MO response, the fixed point problem corresponds to a single SCF iteration where the fixed point is the converged MO. However, in practice, an SCF iteration generally alters the phase of the MO, which violates the assumption that the MO is a fixed point. As such, naively applying AD to differentiate such a problem will lead to the wrong derivatives. The remedy can be simply to enforce the use of a consistent converged MO throughout the gradient propagation, which effectively is equivalent to fixing the gauge.

III. APPLICATIONS

In this section, we provide a few examples that demonstrate the capabilities of PySCFAD. In addition, complete code snippets are presented to highlight the ease of developing new methodologies within the framework of PySCFAD. PySCFAD allows both forward- and reverse-mode AD for its functions. Recall that forward- and reverse-mode AD have time complexities that scale linearly with the numbers of variables and objectives, respectively. It is computationally more efficient to use reverse-mode AD when there are more variables than objectives. As such, reverse-mode AD was applied in all the following calculations for better performance.

A. Orbital optimization

Orbital optimization in electronic structure methods provides the following advantages:

1. Energies become variational with respect to orbital rotations; thus, there is no need for the orbital response when computing nuclear gradients.[30]
2. Properties can be computed more easily because there are no orbital response contributions to the density matrices.
3. The spurious poles in response functions for inexact methods such as coupled cluster theory can be removed.[31]
4. Symmetry-breaking problems may be better described.[32]

However, it is not always straightforward to implement analytic orbital gradients (and hessians for quadratically convergent optimization) if the underlying electronic structure method is complicated. In contrast, little effort is needed to obtain the orbital gradients and hessians using AD as long as the energy can be defined and implemented as differentiable with respect to orbital rotations.

As an example, in Fig. 2, we show the application of orbital optimization to the random phase approximation for the energy[33–35] (RPA) within the framework of PySCFAD. The base RPA method is implemented following Ren *et al.*,[36] where the correlation energy is evaluated by numerical integration over the imaginary frequency

```python
import numpy
from scipy.optimize import minimize
import jax
from pyscf.df.addons import make_auxbasis
from pyscfad import gto, dft, df
from pyscfad.gw import rpa
from pyscfad.tools import rotate_mo1

# molecular structure
mol = gto.Mole()
mol.atom = [['He', (0., 0., 0.)],
            ['He', (0., 0., 2.6)]]
mol.basis = 'def2-svp'
mol.build()

# RKS/PBE
mf = dft.RKS(mol, xc='PBE')
mf.kernel()
mo_coeff = mf.mo_coeff

# density fitting basis
dfobj = df.DF(mol, make_auxbasis(mol, mp2fit=True))

def rpa_energy(x):
    # apply orbital rotation
    mf.mo_coeff = rotate_mo1(mo_coeff, x)
    # density-fitted RPA
    myrpa = rpa.RPA(mf)
    myrpa.with_df = dfobj
    myrpa.kernel()
    return myrpa.e_tot

# jacobian
jac = lambda x, *args: jax.jacrev(rpa_energy)(x)
# hessian vector product
hessp = lambda x, p, *args: jax.vjp(jac, x)[1](p)[0]

x0 = numpy.zeros([mol.nao*(mol.nao+1)//2,])
res = minimize(rpa_energy, x0, jac=jac, hessp=hessp,
               method='trust-krylov', options={'gtol': 1e-6})
print(f'OO-RPA/PBE energy: {rpa_energy(res.x)}')
```

FIG. 2. Application of orbital optimization for density-fitted RPA within the framework of PySCFAD.

axis and density fitting is applied to reduce the computation cost. The total energy is then minimized with respect to the orbital rotation $e^{\mathbf{x}}$, where $\mathbf{x}$ is anti-Hermitian and its upper triangular part corresponds to the variable "x" in Fig. 2. Note that only the energy function needs to be explicitly implemented (lines 24–31 in Fig. 2), whereas the orbital gradient and hessian (specifically, the hessian-vector product in the example) are obtained directly by AD (lines 33–36 in Fig. 2). The performance of the resulting orbital optimized (OO) RPA method is shown in Fig. 3. We plot the binding energy curves for the He_2 molecule computed at the RPA, OO-RPA, and CCSD(T) levels. It is clear that the molecule is underbound at the RPA level, whereas applying orbital optimization corrects that, and the binding energies are more consistent with the CCSD(T) results.

B. Response properties

In general, ground-state *dynamic* (frequency-dependent) response properties are related to the derivatives of the quasienergy with respect to perturbations.[37] The quasienergy is defined as the time-averaged expectation value of $\hat{H} - i\partial/\partial t$ over the phase-isolated time-dependent wavefunction.[37] As such, although there is no difficulty applying AD to compute such derivatives, a straightforward application to the quasienergy in this form requires the time-dependent wavefunction itself, which may be technically challenging. In the time-independent limit, however, the quasienergy reduces to the usual energy; thus, the *static* response properties can be computed straightforwardly with AD from only time-independent quantities.

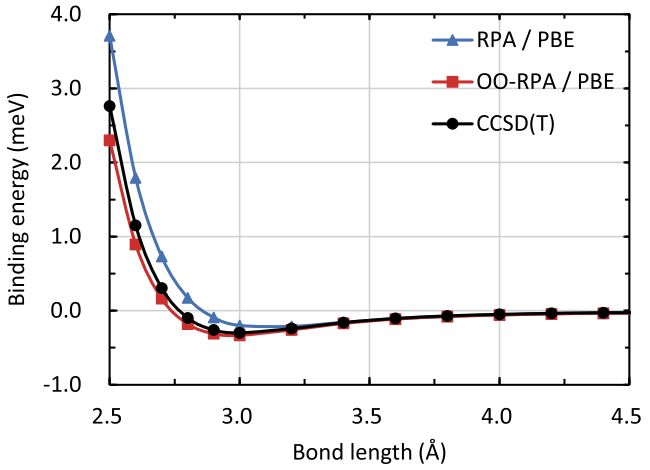


FIG. 3. Binding energy curves for the He_2 molecule computed at the RPA, OO-RPA, and CCSD(T) levels. The def2-SVP basis set was used throughout, and the PBE functional was used for the RPA methods. The curves are shifted to match at a bond length of 10.0 Å.

For example, the static Raman activity is related to the susceptibility[38,39]

$$\chi = \frac{\partial^3 E}{\partial \mathbf{R} \partial \boldsymbol{\varepsilon}^2}, \tag{15}$$

where E is the ground-state energy, $\mathbf{R}$ denotes the nuclear coordinates, and $\boldsymbol{\varepsilon}$ represents the electric field. In Fig. 4, we present an implementation of Raman activity at the CCSD level within the framework of PySCFAD. Again, only the energy function needs to be explicitly implemented (lines 14–24 in Fig. 4), while all the derivatives are obtained by AD (lines 27 and 32 in Fig. 4). In particular, the CPHF and CC Λ equations are solved implicitly through the implicit differentiation procedure. Using this code, we compute the harmonic vibrational frequency, Raman activity, and depolarization ratio for the BH molecule, and the results are displayed in Table I. It can be seen that they agree very well with the reference analytic results, which validate our AD implementation.

Similarly, the IR intensity, which involves computing the nuclear derivative of the dipole moment, can be obtained in the same manner with AD. In Table II, the IR intensities of the H_2O molecule at the CCSD level are displayed. Both AD and analytic evaluation give almost identical results.

C. Excitation energies

Excitation energies can be identified from the poles of response functions and can be obtained by solving the (generalized) eigenvalue problems that arise from the derivatives of quasienergy Lagrangians.[37] For example, in coupled cluster theory, the excitation energies can be computed as the eigenvalues of the CC Jacobian, which is defined as the second-order derivative of the zeroth-order Lagrangian with respect to the amplitudes and to the multipliers,

$$\mathbf{A} = \frac{\partial^2 L^{(0)}}{\partial \mathbf{t}^{(0)} \partial \boldsymbol{\lambda}^{(0)}}. \tag{16}$$

Note that $\frac{\partial L^{(0)}}{\partial \boldsymbol{\lambda}^{(0)}}$ gives the CC amplitude equations, which are already implemented in the ground state CC methods. Therefore, the CC Jacobian and subsequently the excitation energies can be obtained effortlessly if AD is applied to differentiate the amplitude equations. In Fig. 5, we show such an example of computing the excitation energies at the CCSD level, where only the CC amplitude equations are explicitly implemented (line 29 in Fig. 5), while the CC Jacobian is obtained via AD (line 33 in Fig. 5) and is then diagonalized. The resulting excitation energies are identical to those from the equation-of-motion (EOM) method (see Table III). Note that it is also possible to only compute the Jacobian vector product rather than the full Jacobian (similar to line 36 in Fig. 2) so that the diagonalization can be performed with Krylov subspace methods,[41] making larger calculations practical.

The strategy above is in principle applicable to any method, with the caveat that complications may arise depending on the particular ansatz used for representing the wavefunction. In general, the eigenvalue problem being solved has the form

$$\mathbf{E}^{[2]}\mathbf{x} = \omega \mathbf{S}^{[2]}\mathbf{x}, \tag{17}$$

where $\mathbf{E}^{[2]}$ is the second-order derivative of the zeroth-order energy or Lagrangian, and $\mathbf{S}^{[2]}$ may be related to the second-order derivative of the wavefunction overlap, which may or may not be the identity.

D. Derivative coupling

The first-order derivative coupling between two electronic states Ψ_I and Ψ_J is defined as[42,43]

$$\mathbf{d}_{IJ} = \langle \Psi_I | \nabla_{\mathbf{R}} \Psi_J \rangle, \tag{18}$$

where $\mathbf{R}$ denotes the nuclear coordinates. Although derivative couplings are closely related to excited-state nuclear gradients, their analytic derivation and implementation may still be tedious and error-prone, depending on the complexity of the underlying electronic structure methods. However, such calculations can be greatly simplified by applying AD.

In Fig. 6, we give an example of computing the derivative coupling at the full configuration interaction[44] (FCI) level within PySCFAD. It should be noted that only a very simple function is explicitly implemented and then differentiated by AD, which is the wavefunction overlap $\langle \Psi_I(\mathbf{R}_0) | \Psi_J(\mathbf{R}) \rangle$. The $\mathbf{R}_0$ here emphasizes that Ψ_I does not respond to the perturbation. In practice, all the variables corresponding to Ψ_I are held constant when defining the objective function (e.g., "mol," "fcivec," and "mf.mo_coeff" at lines 24–26 in Fig. 6). We also note that one can explicitly construct a Lagrangian that is stationary with respect to the bra wavefunction when computing the derivative coupling analytically.[45] In contrast, by applying AD to directly differentiate the ket wavefunction, our approach removes the complication of deriving such a Lagrangian, and this again shows the ease of using AD to compute complex derivatives.

The strategy above is applicable to any wavefunction method. In Table IV, we compare the derivative couplings computed at

```python
from jax import jacrev
from pyscfad import gto, scf, cc
from pyscfad.lib import numpy as np
from pyscfad.prop.thermo import vib

# molecular structure
mol = gto.Mole()
mol.atom = '''B  ,  0.   0.   0.
              H  ,  0.   0.   1.250594'''
mol.basis = 'aug-cc-pvdz'
mol.build(trace_exp=False, trace_ctr_coeff=False)

# CCSD energy under an external static electric field
def energy(mol, E=None):
    mf = scf.RHF(mol)
    if E is not None:
        # applying the field
        field = np.einsum('x,xij->ij', E, mol.intor('int1e_r'))
        h1 = mf.get_hcore() + field
        mf.get_hcore = lambda *args, **kwargs: h1
    mf.kernel()
    mycc = cc.RCCSD(mf)
    mycc.kernel()
    return mycc.e_tot

# energy hessian
hess = jacrev(jacrev(energy))(mol).coords.coords

# set field strenghth to zero
E0 = np.zeros((3))
# Raman tensor
chi = -jacrev(jacrev(jacrev(energy,1),1),0)(mol, E0).coords

# harmonic analysis
vibration, _, raman = vib.harmonic_analysis(mol, hess,
                                            raman_tensor=chi)
print('Vibrational frequency in cm^-1:')
print(vibration['freq_wavenumber'])
print('Raman activity in A^4/amu:')
print(raman['activity'])
print('Depolarization ration:')
print(raman['depolar_ratio'])
```

FIG. 4. An example of computing harmonic vibrational frequencies and Raman activities within the framework of PySCFAD.

the configuration interaction singles[46] (CIS) and FCI levels. It can be seen that our AD implementation produces exactly the same results as the reference implementation. In addition, second-order derivative couplings can be readily obtained by performing AD on the same wavefunction overlap one more time. Finally, with AD, it is even more straightforward to compute the Hellmann–Feynman part of the derivative coupling (which is translationally invariant,[47,48] unlike the full derivative coupling),

$$\mathbf{h}_{IJ} = \langle \mathbf{C}_I | \nabla_{\mathbf{R}} \mathbf{H} | \mathbf{C}_J \rangle, \tag{19}$$

where $\mathbf{H}$ is the Hamiltonian represented in a certain Hilbert space and $\mathbf{C}$ denotes the eigenvectors (i.e., CI vectors) of $\mathbf{H}$. Here, only the Hamiltonian needs to be differentiated (with the orbital response taken into account), but the CI vectors are treated as constants. In other words, it is not necessary to differentiate the CI

TABLE I. Harmonic vibrational frequency, Raman activity, and depolarization ratio of the BH molecule computed at the CCSD/aug-cc-pVDZ level.

Properties	PySCFAD	Reference[a]
Frequency (cm^{-1})	2336.70	2337.24
Raman activity ($Å^4$/amu)	217.84	215.11
Depolarization ratio	0.56	0.56

[a]Reference values obtained from Ref. 39.

TABLE II. Harmonic vibrational frequencies and IR intensities of the H_2O molecule computed at the CCSD/cc-pVDZ level.

	Frequency (cm^{-1})		IR intensity (km/mol)	
Modes	PySCFAD	Reference[a]	PySCFAD	Reference[a]
1	3844	3846	4.47	4.45
2	1700	1697	56.41	56.15
3	3956	3950	22.64	22.62

[a]Reference values obtained from the computational chemistry comparison and benchmark database.[40]

eigensolver (e.g., the Davidson iterations[49]), which greatly simplifies the problem.

E. Property calculation for solids

The PySCFAD framework also works seamlessly for solid calculations. Here, we give an example of computing the stress tensor to illustrate this.

The stress tensor $\boldsymbol{\sigma}$ is defined as the first-order energy response to the infinitesimal strain deformation[51]

$$\sigma_{\alpha\beta} = \frac{1}{V}\frac{\partial E}{\partial \varepsilon_{\alpha\beta}}\bigg|_{\varepsilon=0}, \tag{20}$$

where E and V are the energy and volume of a unit cell, respectively, and the strain tensor $\boldsymbol{\varepsilon}$ defines the transformation of real-space coordinates $\mathbf{R}$ according to

$$R_\alpha(\boldsymbol{\varepsilon}) = \sum_\beta (\delta_{\alpha\beta} + \varepsilon_{\alpha\beta}) R_\beta(\mathbf{0}), \tag{21}$$

in which α and β denote the Cartesian components. The strain deformation applies to both nuclear coordinates and lattice vectors, which also implicitly affects the derived quantities such as the reciprocal lattice vectors and the unit cell volume.

In Fig. 7, the stress tensor for a two-atom Si primitive cell is computed by AD at the Hartree–Fock level, where a mixed Gaussian and plane wave approach (plane-wave density fitting) is used.[52,53]

```python
import numpy
import jax
from pyscfad import gto, scf, cc

# molecular structure
mol = gto.Mole()
mol.atom = 'H 0. 0. 0.; H 0. 0. 0.74'
mol.basis = '631g*'
mol.build()

# HF calculation
mf = scf.RHF(mol)
mf.kernel()

# CCSD calculation
mycc = cc.RCCSD(mf)
mycc.kernel()

# remove the redundant elements in the amplitudes
vec = mycc.amplitudes_to_vector(mycc.t1, mycc.t2)
# obtain the two electron integrals
eris = mycc.ao2mo(mycc.mo_coeff)

# the actual function being differentiated
def amplitude_equation(mycc, vec, eris):
    # retrieve the amplitudes
    t1, t2 = mycc.vector_to_amplitudes(vec)
    # CCSD amplitude equations
    e1, e2 = mycc.amplitude_equation(t1, t2, eris)
    return mycc.amplitudes_to_vector(e1, e2)

# CC Jacobian computed by AD of the amplitude equations
jac = jax.jacfwd(amplitude_equation, 1)(mycc, vec, eris)
# diagonalize the CC Jacobian to get the excitation energies
w, x = numpy.linalg.eig(jac)
print(f'EOM-EE-CCSD singlet state energies: {numpy.sort(w)}')
```

FIG. 5. An example of computing the CC Jacobian by differentiating the amplitude equations within the framework of PySCFAD.

TABLE III. Excitation energies (in eV) for the lowest four singlet excited states of the H_2 molecule (with a bond length of 1.1 Å) computed at the EOM-EE-CCSD/6-31G* level.

States	PySCFAD[a]	Reference[b]
S_1	0.463	0.463
S_2	0.735	0.735
S_3	1.096	1.096
S_4	1.152	1.152

[a]Excitation energies computed by diagonalizing the CC Jacobian from AD.
[b]Reference values computed by the EOM-EE-CCSD method.

TABLE IV. First-order derivative couplings (in a.u.) between the S_1 and S_3 states (both S_1 and S_3 states have single excitation characters, and the S_3 state corresponds to the S_4 state in FCI calculations) for the H_2 molecule (with a bond length of 1.1 Å) computed at the CIS and FCI levels with the cc-pVDZ basis set.

Methods	PySCFAD	Reference
CIS	0.0796	0.0796[a]
FCI	0.2126	0.2126[b]

[a]Reference values computed analytically by Q-Chem.[50]
[b]Reference values computed by finite difference.

The results are plotted in Fig. 8 along with the corresponding finite difference values. Perfect agreement is obtained with an average discrepancy of 4×10^{-8} eV/Å^3 between the AD and finite difference results. It should be noted that, in this case, the plane wave basis (i.e., the uniform grid point) response to the strain deformation is non-negligible, which subsequently requires differentiation with respect to the grid points. However, all these complications are hidden from the user, and it suffices to modify the energy function (as shown in Fig. 7) for the gradient calculations.

IV. COMPUTATIONAL EFFICIENCY

In this section, we investigate the computation cost for carrying out typical quantum chemistry calculations within the PySCFAD framework.

```python
import jax
from pyscfad import gto, scf, fci

# molecular structure
mol = gto.Mole()
mol.atom = 'H 0 0 0; H  0 0 1.1'
mol.basis = 'ccpvdz'
mol.build()

# HF and FCI calculations
mf = scf.RHF(mol)
mf.kernel()
nroots = 8
e, fcivec = fci.solve_fci(mf, nroots=nroots)

nelec = mol.nelectron
norb = mf.mo_coeff.shape[-1]
stateI, stateJ = 2, 7
def ovlp(mol1):
    mf1 = scf.RHF(mol1)
    mf1.kernel()
    e1, fcivec1 = fci.solve_fci(mf1, nroots=nroots)
    # wavefunction overlap
    s = fci.fci_ovlp(mol, mol1, fcivec[stateI], fcivec1[stateJ],
                     norb, norb, nelec, nelec,
                     mf.mo_coeff, mf1.mo_coeff)
    return s

# Only the ket state is differentiated
mol1 = mol.copy()
jac = jax.jacrev(ovlp)(mol1)
print("FCI derivative coupling:")
print(jac.coords)
```

FIG. 6. An example of computing the derivative coupling between FCI wavefunctions within the framework of PySCFAD.

```python
from jax import value_and_grad
from pyscf.data.nist import BOHR, HARTREE2EV
from pyscfad.lib import numpy as np
from pyscfad.pbc import gto as pbcgto
from pyscfad.pbc import scf as pbcscf

def hf_energy(strain, a):
    cell = pbcgto.Cell()
    cell.atom = [['Si', [0., 0., 0.]],
                 ['Si', [a/4, a/4, a/4]]]
    cell.a = np.asarray([[0., a/2, a/2],
                         [a/2, 0., a/2],
                         [a/2, a/2, 0.]])
    cell.basis = 'gth-szv'
    cell.pseudo = 'gth-pade'
    cell.exp_to_discard = 0.1
    cell.build(trace_lattice_vectors=True)
    # apply strain to lattice vectors
    cell.abc += np.einsum('ab,nb->na', strain,
                          cell.lattice_vectors())
    # apply strain to nuclear coordinates
    cell.coords += np.einsum('xy,ny->nx', strain,
                             cell.atom_coords())

    kpts = cell.make_kpts([2, 2, 2])
    mf = pbcscf.KRHF(cell, kpts=kpts)
    ehf = mf.kernel()
    return ehf, cell.vol

strain = np.zeros((3,3))
(ehf, vol), jac = value_and_grad(hf_energy,
                                 has_aux=True)(strain, 5.431)
print('stress tensor in eV/A^3:')
print(jac * HARTREE2EV / (vol * (BOHR ** 3)))
```

FIG. 7. An example of computing the stress tensor within the framework of PySCFAD. The system being studied is a two-atom primitive cell of Si with a lattice parameter of 5.431 Å. The total energy is computed at the HF level using the GTH-SZV basis set and the GTH-PADE pseudopotential.[54]

First, we consider objective function evaluation. As an example, in Fig. 9, we plot the computation time[55] for computing the CCSD(T) energy of the H_2O molecule using the implementations in PySCF and PySCFAD, respectively. The main difference between the two implementations is that PySCF respects the eight-fold permutation symmetry of the two-electron repulsion integrals, while no such symmetry is enforced in PySCFAD. For PySCFAD, we also compare the performance of applying Jax and NumPy as the tensor operation backends, respectively. The corresponding results are presented as the red (PySCF), blue and orange (PySCFAD with Jax), and gray (PySCFAD with NumPy) bars in Fig. 9. We see that PySCFAD with the Jax backend is about 5 ~ 8 times less efficient than PySCF, which is due to the lack of integral symmetries. However, PySCFAD with the NumPy backend performs much worse for larger calculations. This is mainly because the *einsum* function in NumPy, when applied to general tensor contractions, may be less optimized and only runs on a single core. With the Jax backend, the same function is optimized by XLA[56] (accelerated linear algebra), which greatly improves the computation efficiency, although the just-in-time compilation introduces some overhead, as shown by the orange bars in Fig. 9. Overall, this suggests that it is potentially possible for PySCFAD to be just as efficient as PySCF for objective function evaluation if the implementation is carefully optimized.

Second, we examine the efficiency of AD for derivative calculations. In Fig. 10, the computation time[55] for CCSD nuclear

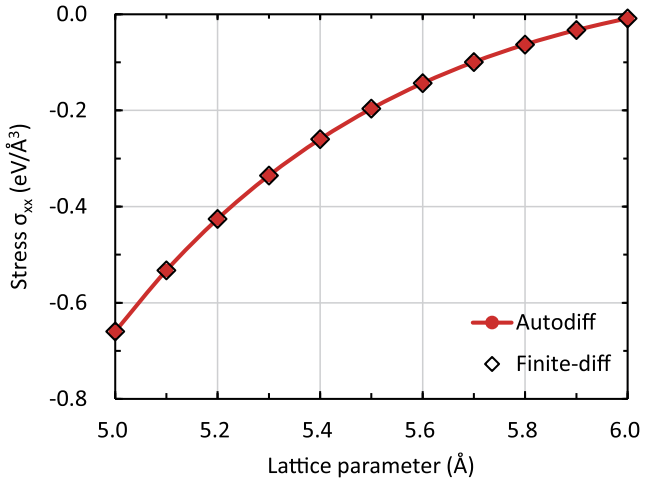


FIG. 8. Stress tensor component σ_{xx} for two-atom Si primitive cells computed at the HF level with the GTH-SZV basis and GTH-PADE pseudopotential. For various lattice parameters, the results by AD (red) and by finite difference (black) are plotted.

gradients of the H_2O molecule using the analytic implementation in PySCF and the AD procedures in PySCFAD is plotted. Note that two iterative solvers are involved in the CCSD energy evaluation, i.e., the SCF iteration and the CC amplitude equation. When applying AD to compute the gradient, one can choose to use either the approach of unrolling the iterations or the implicit differentiation of the two iterative solvers. As such, the performance of these AD approaches is also compared, as displayed by the three sets of blue bars in Fig. 10. It is clear that all the AD calculations

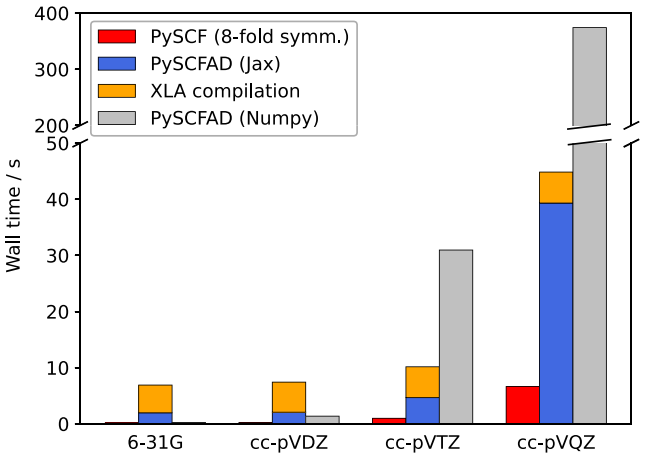


FIG. 9. Wall time of CCSD(T) energy calculations for the H_2O molecule with various basis sets, using different implementations. PySCF gives results that are displayed as red bars. The blue-and-orange bars represent the total wall time using PySCFAD, where the orange and blue portions correspond to the XLA compilation time and the remaining code execution time, respectively. The gray bars show the results from the NumPy version of the PySCFAD implementation (where all the Jax functions are replaced with their NumPy counterparts). The calculations were run using 16 Intel Xeon E5-2697 v4 @ 2.30 GHz core.

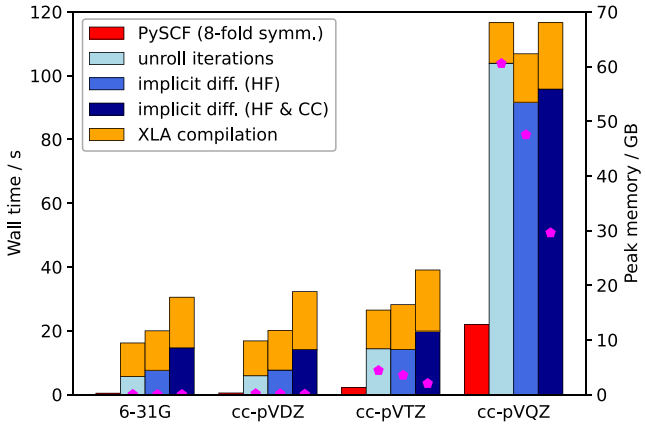


FIG. 10. Wall time of CCSD nuclear gradient calculations for the H_2O molecule with various basis sets, using different implementations. PySCF gives results that are displayed as red bars. The blue-and-orange bars represent the total wall time using PySCFAD, where the orange and blue portions correspond to the XLA compilation time and the remaining code execution time, respectively. Three AD strategies for evaluating the derivatives of iterative solvers are investigated, namely, unrolling the iterations (light blue), implicit differentiation of the SCF iterations (blue), and implicit differentiation of both the SCF iterations and the CC amplitude equations (dark blue). The memory footprint of the AD calculations is labeled by the magenta pentagons. The calculations were run using 16 Intel Xeon E5-2697 v4 @ 2.30 GHz cores.

are less efficient than the analytic evaluations by a factor of 5 ~ 8, which is again due to the lack of integral symmetries. On the other hand, the different AD approaches give comparable performance in terms of computation time. However, implicit differentiation of the iterative solvers consumes a considerably smaller amount of memory than unrolling the iterations. In more detail, for the case of the cc-pVQZ basis set in Fig. 10, more than 20 gigabytes of memory are used to store the intermediate quantities for computing the gradients of the ERIs, which applies equally to all the AD approaches. The remaining memory can be associated with the iterative solvers, and we find that implicit differentiation of the SCF and CC iterations reduces the associated memory usage by a factor of four compared to the approach of unrolling the iterations. In conclusion, PySCFAD appears to be computationally efficient for derivative evaluations, especially when the iterative solvers are differentiated implicitly.

V. CONCLUSIONS

In this work, we introduced PySCFAD, a differentiable quantum chemistry framework based on PySCF. It facilitates derivative calculations for complex computational workflows and can be applied to both molecular and periodic systems, at the mean-field level and beyond. Using PySCFAD, new quantum chemistry methods can be implemented in a differentiable way with almost no extra effort. As such, we expect it to become a useful platform to rapidly prototype new methodologies, which can then be benchmarked on properties that were previously difficult to compute due to the lack of analytic derivatives. At the current point, there remain a few challenges to be addressed in the future development of PySCFAD.

1. JAX is less efficient than NUMPY unless the Python functions are compiled with XLA. However, this is not always possible, especially when Python control flows are involved. As such, code optimization becomes more difficult for PySCFAD.
2. Incorporating permutation symmetries in electron integrals is tricky because it requires element-wise in-place array updates, which are extremely inefficient with current AD tools.
3. The current implementation amounts to an "in-core" version of all methods, i.e., all the data are stored in memory. For "out-core" implementations where data can be stored on disk, additional work is needed to track the history of the data, without which gradient propagation cannot proceed.

ACKNOWLEDGMENTS

This work was supported by the US Department of Energy through the US DOE, Office of Science, Basic Energy Sciences, Chemical Sciences, Geosciences, and Biosciences Division, under Triad National Security, LLC ("Triad") contract Grant No. 89233218CNA000001. Support for the PySCF ML infrastructure on top of which PySCFAD was built comes from the Dreyfus Foundation.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Xing Zhang: Conceptualization (equal); Methodology (lead); Software (lead); Validation (equal); Writing – original draft (lead). **Garnet Kin-Lic Chan**: Conceptualization (equal); Funding acquisition (lead); Project administration (lead); Resources (lead); Supervision (lead); Validation (equal); Writing – review & editing (lead).

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request. The PySCFAD source code can be found at https://github.com/fishjojo/pyscfad.

REFERENCES

[1] R. E. Wengert, "A simple automatic derivative evaluation program," Commun. ACM **7**, 463–464 (1964).
[2] A. Griewank and A. Walther, *Evaluating Derivatives* (Society for Industrial and Applied Mathematics, 2008).
[3] T. Tamayo-Mendoza, C. Kreisbeck, R. Lindh, and A. Aspuru-Guzik, "Automatic differentiation in quantum chemistry with applications to fully variational Hartree–Fock," ACS Cent. Sci. **4**, 559–566 (2018).
[4] C. Song, T. J. Martínez, and J. B. Neaton, "An automatic differentiation and diagrammatic notation approach for developing analytical gradients of tensor hyper-contracted electronic structure methods," ChemRxiv:13002965.v1 (2020).
[5] A. S. Abbott, B. Z. Abbott, J. M. Turney, and H. F. Schaefer, "Arbitrary-order derivatives of quantum chemical methods via automatic differentiation," J. Phys. Chem. Lett. **12**, 3232–3239 (2021).

[6]M. F. Kasim, S. Lehtola, and S. M. Vinko, "DQC: A python program package for differentiable quantum chemistry," J. Chem. Phys. **156**, 084801 (2022).
[7]L. Li, S. Hoyer, R. Pederson, R. Sun, E. D. Cubuk, P. Riley, and K. Burke, "Kohn-Sham equations as regularizer: Building prior knowledge into machine-learned physics," Phys. Rev. Lett. **126**, 036401 (2021).
[8]M. F. Kasim and S. M. Vinko, "Learning the exchange-correlation functional from nature with fully differentiable density functional theory," Phys Rev Lett. **127**, 126403 (2021).
[9]H.-J. Liao, J.-G. Liu, L. Wang, and T. Xiang, "Differentiable programming tensor networks," Phys. Rev. X **9**, 031041 (2019).
[10]J. M. Arrazola, S. Jahangiri, A. Delgado, J. Ceroni, J. Izaac, A. Száva, U. Azad, R. A. Lang, Z. Niu, O. D. Matteo, R. Moyard, J. Soni, M. Schuld, R. A. Vargas-Hernández, T. Tamayo-Mendoza, C. Y.-Y. Lin, A. Aspuru-Guzik, and N. Killoran, "Differentiable quantum computational chemistry with pennylane," arXiv:2111.09967.
[11]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, "PySCF: The python-based simulations of chemistry framework," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **8**, e1340 (2018).
[12]Q. Sun, X. Zhang, S. Banerjee, P. Bao, M. Barbry, N. S. Blunt, N. A. Bogdanov, G. H. Booth, J. Chen, Z.-H. Cui, J. J. Eriksen, Y. Gao, S. Guo, J. Hermann, M. R. Hermes, K. Koh, P. Koval, S. Lehtola, Z. Li, J. Liu, N. Mardirossian, J. D. McClain, M. Motta, B. Mussard, H. Q. Pham, A. Pulkin, W. Purwanto, P. J. Robinson, E. Ronca, E. R. Sayfutyarova, M. Scheurer, H. F. Schurkus, J. E. T. Smith, C. Sun, S.-N. Sun, S. Upadhyay, L. K. Wagner, X. Wang, A. White, J. D. Whitfield, M. J. Williamson, S. Wouters, J. Yang, J. M. Yu, T. Zhu, T. C. Berkelbach, S. Sharma, A. Y. Sokolov, and G. K.-L. Chan, "Recent developments in the PySCF program package," J. Chem. Phys. **153**, 024109 (2020).
[13]Dataset: X. Zhang, J. Yang, and G. K.-L. Chan (2022). "PySCF with auto-differentiation," Zenodo.
[14]C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," Nature **585**, 357–362 (2020).
[15]P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, A. Vijaykumar, A. P. Bardelli, A. Rothberg, A. Hilboll, A. Kloeckner, A. Scopatz, A. Lee, A. Rokem, C. N. Woods, C. Fulton, C. Masson, C. Häggström, C. Fitzgerald, D. A. Nicholson, D. R. Hagen, D. V. Pasechnik, E. Olivetti, E. Martin, E. Wieser, F. Silva, F. Lenders, F. Wilhelm, G. Young, G. A. Price, G.-L. Ingold, G. E. Allen, G. R. Lee, H. Audren, I. Probst, J. P. Dietrich, J. Silterra, J. T. Webber, J. Slavič, J. Nothman, J. Buchner, J. Kulick, J. L. Schönberger, J. V. de Miranda Cardoso, J. Reimer, J. Harrington, J. L. C. Rodríguez, J. Nunez-Iglesias, J. Kuczynski, K. Tritz, M. Thoma, M. Newville, M. Kümmerer, M. Bolingbroke, M. Tartre, M. Pak, N. J. Smith, N. Nowaczyk, N. Shebanov, O. Pavlyk, P. A. Brodtkorb, P. Lee, R. T. McGibbon, R. Feldbauer, S. Lewis, S. Tygier, S. Sievert, S. Vigna, S. Peterson, S. More, T. Pudlik, T. Oshima, T. J. Pingel, T. P. Robitaille, T. Spura, T. R. Jones, T. Cera, T. Leslie, T. Zito, T. Krauss, U. Upadhyay, Y. O. Halchenko, and Y. Vázquez-Baeza, "SciPy 1.0: Fundamental algorithms for scientific computing in python," Nat. Methods **17**, 261–272 (2020).
[16]S. F. Walter and L. Lehmann, "Algorithmic differentiation in python with AlgoPy," J. Comput. Sci. **4**, 334–344 (2013).
[17]A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. D. Facebook, A. I. Research, Z. Lin, A. Desmaison, L. Antiga, O. Srl, and A. Lerer, "Automatic differentiation in PyTorch," NIPS 2017 Autodiff Workshop (2017); available at http://openreview.net/forum?id=BJJsrmfCZ.
[18]Dataset: J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang (2018). "JAX: Composable transformations of Python + NumPy programs," Github. http://github.com/google/jax.
[19]Q. Sun, "Libcint: An efficient general integral library for Gaussian basis functions," J. Comput. Chem. **36**, 1664–1671 (2015).
[20]W. Moses and V. Churavy, "Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients," in *Advances in Neural Information Processing System*, edited by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Curran Associates, Inc., 2020), Vol. 33, pp. 12472–12485.
[21]S. Lehtola, C. Steigemann, M. J. T. Oliveira, and M. A. L. Marques, "Recent developments in LIBXC—A comprehensive library of functionals for density functional theory," SoftwareX **7**, 1–5 (2018).
[22]J. A. Pople, R. Krishnan, H. B. Schlegel, and J. S. Binkley, "Derivative studies in Hartree-Fock and Møller-Plesset theories," Int. J. Quantum Chem. **16**, 225–241 (1979).
[23]P. Ablin, G. Peyré, and T. Moreau, "Super-efficiency of automatic differentiation for functions defined as a minimum," in *International Conference on Machine Learning* (PMLR, 2020), pp. 32–41.
[24]J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A software framework for nonlinear optimization and optimal control," Math. Program. Comput. **11**, 1–36 (2019).
[25]M. Blondel, Q. Berthet, M. Cuturi, R. Frostig, S. Hoyer, F. Llinares-López, F. Pedregosa, and J.-P. Vert, "Efficient and modular implicit differentiation," arXiv:2105.15183.
[26]S. G. Krantz and H. R. Parks, *The Implicit Function Theorem: History, Theory, and applications* (Springer Science and Business Media, 2002).
[27]N. C. Handy and H. F. Schaefer, "On the evaluation of analytic energy derivatives for correlated wave functions," J. Chem. Phys. **81**, 5031–5033 (1984).
[28]P. Pulay, "Improved SCF convergence acceleration," J. Comput. Chem. **3**, 556–560 (1982).
[29]A. C. Scheiner, G. E. Scuseria, J. E. Rice, T. J. Lee, and H. F. Schaefer, "Analytic evaluation of energy gradients for the single and double excitation coupled cluster (CCSD) wave function: Theory and application," J. Chem. Phys. **87**, 5361–5373 (1987).
[30]A. P. L. Rendell, G. B. Bacskay, N. S. Hush, and N. C. Handy, "The analytic configuration interaction gradient method: The calculation of one electron properties," J. Chem. Phys. **87**, 5976–5986 (1987).
[31]T. B. Pedersen, B. Fernández, and H. Koch, "Gauge invariant coupled cluster response theory using optimized nonorthogonal orbitals," J. Chem. Phys. **114**, 6983–6993 (2001).
[32]C. D. Sherrill, A. I. Krylov, E. F. Byrd, and M. Head-Gordon, "Energies and analytic gradients for a coupled-cluster doubles model using variational Brueckner orbitals: Application to symmetry breaking in O_4^+," J. Chem. Phys. **109**, 4171 (1998).
[33]D. Bohm and D. Pines, "A collective description of electron interactions: III. Coulomb interactions in a degenerate electron gas," Phys. Rev. **92**, 609–625 (1953).
[34]M. Gell-Mann and K. A. Brueckner, "Correlation energy of an electron gas at high density," Phys. Rev. **106**, 364–368 (1957).
[35]F. Furche, "Molecular tests of the random phase approximation to the exchange-correlation energy functional," Phys. Rev. B **64**, 195120 (2001).
[36]X. Ren, P. Rinke, V. Blum, J. Wieferink, A. Tkatchenko, A. Sanfilippo, K. Reuter, and M. Scheffler, "Resolution-of-identity approach to Hartree–Fock, hybrid density functionals, RPA, MP2 and *GW* with numeric atom-centered orbital basis functions," New J. Phys. **14**, 053020 (2012).
[37]O. Christiansen, P. Jørgensen, and C. Hättig, "Response functions from fourier component variational perturbation theory applied to a time-averaged quasienergy," Int. J. Quantum Chem. **68**, 1–52 (1998).
[38]G. Placzek, "Rayleigh-Streuung und Raman-Effekt," *Hanbuch der Radiologie* (Akademische, 1934), pp. 205–374.
[39]D. P. O'Neill, M. Kállay, and J. Gauss, "Analytic evaluation of Raman intensities in coupled-cluster theory," Mol. Phys. **105**, 2447–2453 (2007).
[40]R. D. Johnson III, Nist computational chemistry comparison and benchmark database, NIST standard reference database number 101, Release 16a http://cccbdb.nist.gov/; accessed 13 March 2015) (2013).
[41]Y. Saad, *Numerical Methods for Large Eigenvalue Problems: Revised Edition* (SIAM, 2011).
[42]J. C. Tully, "Molecular dynamics with electronic transitions," J. Chem. Phys. **93**, 1061–1071 (1990); arXiv:1011.1669v3.

[43]D. R. Yarkony, "Diabolical conical intersections," Rev. Mod. Phys. **68**, 985–1013 (1996).
[44]P. J. Knowles and N. C. Handy, "A new determinant-based full configuration interaction method," Chem. Phys. Lett. **111**, 315–321 (1984).
[45]E. G. Hohenstein, "Analytic formulation of derivative coupling vectors for complete active space configuration interaction wavefunctions with floating occupation molecular orbitals," J. Chem. Phys. **145**, 174110 (2016).
[46]D. Maurice and M. Head-gordon, "Configuration interaction with single substitutions for excited," Int. J. Quantum Chem. **56**, 361–370 (1995).
[47]S. Fatehi, E. Alguire, Y. Shao, and J. E. Subotnik, "Analytic derivative couplings between configuration-interaction-singles states with built-in electron-translation factors for translational invariance," J. Chem. Phys. **135**, 234105 (2011).
[48]X. Zhang and J. M. Herbert, "Analytic derivative couplings for spin-flip configuration interaction singles and spin-flip time-dependent density functional theory," J. Chem. Phys. **141**, 064104–064109 (2014).
[49]E. R. Davidson, "The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices," J. Comput. Phys. **17**, 87–94 (1975).
[50]E. Epifanovsky, A. T. Gilbert, X. Feng, J. Lee, Y. Mao, N. Mardirossian, P. Pokhilko, A. F. White, M. P. Coons, A. L. Dempwolff *et al.*, "Software for the frontiers of quantum chemistry: An overview of developments in the Q-Chem 5 package," J. Chem. Phys. **155**, 084801 (2021).
[51]F. Knuth, C. Carbogno, V. Atalla, V. Blum, and M. Scheffler, "All-electron formalism for total energy strain derivatives and stress tensor components for numeric atom-centered orbitals," Comput. Phys. Commun. **190**, 33–50 (2015).
[52]J. VandeVondele, M. Krack, F. Mohamed, M. Parrinello, T. Chassaing, and J. Hutter, "QUICKSTEP: Fast and accurate density functional calculations using a mixed Gaussian and plane waves approach," Comput. Phys. Commun. **167**, 103–128 (2005).
[53]J. McClain, Q. Sun, G. K.-L. Chan, and T. C. Berkelbach, "Gaussian-based coupled-cluster theory for the ground-state and band structure of solids," J. Chem. Theory Comput. **13**, 1209–1218 (2017).
[54]S. Goedecker, M. Teter, and J. Hutter, "Separable dual-space Gaussian pseudopotentials," Phys. Rev. B **54**, 1703–1710 (1996); arXiv:9512004 [mtrl-th].
[55]The reported times are for the entire calculations, *i.e.*, including electron integral evaluations, SCF iterations, CC energies, and (for Fig. 10) nuclear gradients.
[56]Dataset: A. Sabne (2020). "XLA: Compiling machine learning for peak performance," Google Research. http://research.google/pubs/pub50530.