

DQC: a Python program package for Differentiable Quantum Chemistry

M. F. Kasim,^{1,*} S. Lehtola,² and S. M. Vinko^{1,3,†}

¹*Department of Physics, Clarendon Laboratory, University of Oxford, Parks Road, Oxford OX1 3PU, UK*

²*Molecular Sciences Software Institute, Blacksburg, Virginia 24061, United States*

³*Central Laser Facility, STFC Rutherford Appleton Laboratory, Didcot OX11 0QX, UK*

(Dated: February 7, 2022)

Automatic differentiation represents a paradigm shift in scientific programming, where evaluating both functions and their derivatives is required for most applications. By removing the need to explicitly derive expressions for gradients, development times can be shortened, and calculations simplified. For these reasons, automatic differentiation has fueled the rapid growth of a variety of sophisticated machine learning techniques over the past decade, but is now also increasingly showing its value to support *ab initio* simulations of quantum systems, and enhance computational quantum chemistry. Here we present an open-source differentiable quantum chemistry simulation code, DQC, and explore applications facilitated by automatic differentiation: (1) calculating molecular perturbation properties; (2) reoptimizing a basis set for hydrocarbons; (3) checking the stability of self-consistent field wave functions; and (4) predicting molecular properties via alchemical perturbations.

I. INTRODUCTION

Automatic differentiation is a collection of techniques used to evaluate, up to machine precision, the derivative of a function specified by a computer program. It allows software developers to focus solely on designing the best model for a given problem, without having to worry about implementing any derivatives of the model with respect to its various mathematical parameters. It has already had a transformative effect in machine learning, enabling the development of many new techniques over the past decade, such as batch normalization [1], attention layers [2], and unique neural network architectures [3, 4].

Automatic differentiation is still relatively new in the context of computational sciences, but is already showing promise across a diverse set of applications including tensor networks [5], computational fluid dynamics [6], and molecular dynamics simulations [7]. Automatic differentiation is also growing increasingly popular in quantum chemistry, where it has been used to optimize molecular basis sets [8], to calculate higher derivatives of various exchange-correlation (xc) functionals [9] of density functional theory (DFT) [10, 11], and to determine arbitrary-order nuclear coordinate derivatives of electronic energies [12].

Automatic differentiation is also an essential stepping stone to enable direct integration of quantum chemistry methods with machine learning models and their training. In this context, a differentiable implementation of DFT was recently used to learn the xc functional [13] from accurate reference calculations within the density matrix renormalization group approach, or from a mixture of computational and experimental data [14], show-

ing a promising new approach to developing transferable and robust xc functionals via deep learning.

Although differentiation in quantum chemistry can be done via finite-difference schemes, calculating the gradient of a parameter with respect to some other input parameter can be time consuming as one has to run the simulation as many times as the number of input parameters. Moreover, finite-difference methods are prone to numerical instabilities and are very sensitive to the chosen step-size. Automatic differentiation addresses these challenges by calculating the analytical gradient via the chain rule, eliminating both the need to run the simulation many times, and the need for step-size tuning.

To address the growing need for automatic differentiation in quantum chemistry, we introduce Differentiable Quantum Chemistry (DQC), a DFT and Hartree-Fock (HF) [15] simulation code. DQC is written in Python using PyTorch [16] and xitorch [17]. While PyTorch provides gradient calculations for elementary operations such as matrix multiplication and explicit eigendecomposition, xitorch provides gradient calculations for functional operations such as root finding, optimization, and implicit eigendecomposition. The use of PyTorch and xitorch in DQC enables various applications in quantum chemistry that would otherwise be far more difficult to pursue. For example, the work by Kasim and Vinko [14] on learning the xc functional directly from a set of heterogeneous experimental data and calculated density profiles within the constraints of Kohn-Sham DFT already mentioned above was enabled by the use of DQC.

We begin this paper by outlining the basic theory behind DQC in section II. The implementation is described in section III. Applications that exemplify the present approach are given in section IV. We discuss the challenges in the use of automatic differentiation techniques in computational science in section V, and summarize our work in section VI.

* Present address: Machine Discovery Ltd, muhammad@machine-discovery.com

† sam.vinko@physics.ox.ac.uk

II. METHODS

Quantum chemical calculations of the electronic structure typically require the evaluation of abstract functionals, such as root finding for self-consistent field (SCF) iterations, and minimization for geometry optimizations and the direct minimization approach to the SCF problem. We discuss the handling of these functionals in the context of DQC in this section.

A. SCF iterations

DQC is based on the use of a Gaussian basis set within the linear combination of atomic orbitals approach (LCAO). For simplicity, we will only present the theory for the spin-restricted formalism, as the spin-unrestricted (and restricted open-shell) formalisms are analogous. The SCF iterations proceed in the LCAO approach by solving Roothaan's equation [18, 19]

$$\mathbf{F}(\mathbf{D})\mathbf{C} = \mathbf{S}\mathbf{C}\mathbf{E}, \quad (1)$$

where $\mathbf{F}(\mathbf{D})$ is the Fock matrix which is a function of the density matrix \mathbf{D} , \mathbf{C} is the orbital matrix, \mathbf{S} is the overlap matrix, and \mathbf{E} is a diagonal matrix containing the orbital energies. The generalized eigenvalue equation in equation (1) can be reduced to the normal form by orthogonalizing the overlap matrix \mathbf{S} [20], and operating in the linearly independent basis spanned by the eigenvectors of \mathbf{S} with large enough eigenvalues; any ill-conditioning in the overlap matrix can be removed by choosing a well-defined sub basis with the help of a pivoted Cholesky decomposition [21]. The density matrix \mathbf{D} , required to construct the Fock matrix \mathbf{F} , can be obtained by

$$\mathbf{D} = \mathbf{C}\mathbf{N}\mathbf{C}^\dagger \quad (2)$$

with \mathbf{N} a diagonal matrix containing the occupation numbers of the orbitals. As discussed in ref. 19, the SCF procedure requires repeatedly solving equations (1) and (2) until self-consistency is achieved.

The Roothaan iteration in equation (1) can be written mathematically as the process of finding a vector \mathbf{y} such that

$$\mathbf{y} = \mathbf{f}(\mathbf{y}, \boldsymbol{\theta}), \quad (3)$$

where \mathbf{f} is a function that takes the vector \mathbf{y} and other parameters $\boldsymbol{\theta}$, and returns the expected value of vector \mathbf{y} . In DQC, the parameter \mathbf{y} is represented by the Fock matrix \mathbf{F} , so the function \mathbf{f} is the procedure that solves equation (1), calculating the density matrix from equation (2) and constructing back the Fock matrix from the density matrix. The parameters $\boldsymbol{\theta}$ represent the other variables involved in the calculation, such as the overlap matrix \mathbf{S} and the occupation number matrix \mathbf{N} . The algorithm and gradient calculation for equations (1) and (3) are already available in PyTorch and xitorch.

TABLE I. Execution speed comparison between DQC (SCF iterations) and PySCF.

Cases	DQC	PySCF
H ₂ O (HF/cc-pVDZ)	96 ms	245 ms
H ₂ O (PW92/cc-pVDZ)	530 ms	430 ms
C ₄ H ₅ N (HF/cc-pVTZ)	108 s	17 s
C ₄ H ₅ N (PW92/cc-pVTZ)	101 s	25 s
C ₄ H ₅ N (density fit PW92/cc-pVTZ)	30 s	22 s
C ₆ H ₈ O ₆ (density fit PW92/cc-pVDZ)	87 s	57 s

B. Direct minimization

An alternative approach to solving the self-consistent field equations is to directly find the orbitals that minimize the total energy \mathcal{E} by the use of optimization algorithms [22]. The energy \mathcal{E} can be calculated from the density matrix \mathbf{D} which can be obtained in turn from the orbital coefficients \mathbf{C} using equation (2). This relation makes the energy a function of the orbital matrix, $\mathcal{E}(\mathbf{C})$. However, as the orbitals must remain orthonormal, $\mathbf{C}^\dagger\mathbf{S}\mathbf{C} = \mathbf{I}$, we introduce a new variable \mathbf{Q} , defined in terms of its relation with \mathbf{C} as

$$\mathbf{Q} = \hat{\mathbf{Q}}\mathbf{R} \quad (4)$$

$$\mathbf{C} = \mathbf{S}^{-1/2}\hat{\mathbf{Q}}. \quad (5)$$

Equation (4) is the QR decomposition of the matrix \mathbf{Q} into an orthogonal matrix $\hat{\mathbf{Q}}$ and an upper triangular matrix \mathbf{R} . Equation (5) involves the inverse square root of the overlap matrix \mathbf{S} , which can be computed using the eigendecomposition of \mathbf{S} . The energy can then be parameterized by \mathbf{Q} , reducing the direct minimization problem to an unbounded optimization problem:

$$\mathbf{Q}^* = \arg \min_{\mathbf{Q}} \mathcal{E}(\mathbf{Q}). \quad (6)$$

The gradient of the energy with respect to \mathbf{Q} , i.e. $\partial\mathcal{E}/\partial\mathbf{Q}$, is required for an efficient solution. It is automatically computed by PyTorch and xitorch.

Once the optimum \mathbf{Q}^* in equation (6) is found, \mathbf{Q}^* can still be differentiated with respect to any other variables, such as the nuclear coordinates or the occupation number matrix \mathbf{N} . This is made possible by the gradient of the optimization functional provided by xitorch.

We note that our approach of using QR decomposition is slightly different from common direct minimization techniques that use the matrix exponential of a skew-Hermitian matrix such as the geometric direct minimization algorithm of ref. 23.

C. Automatic differentiation with PyTorch

PyTorch [16] is a dynamic automatic differentiation library written in Python that provides gradients au-

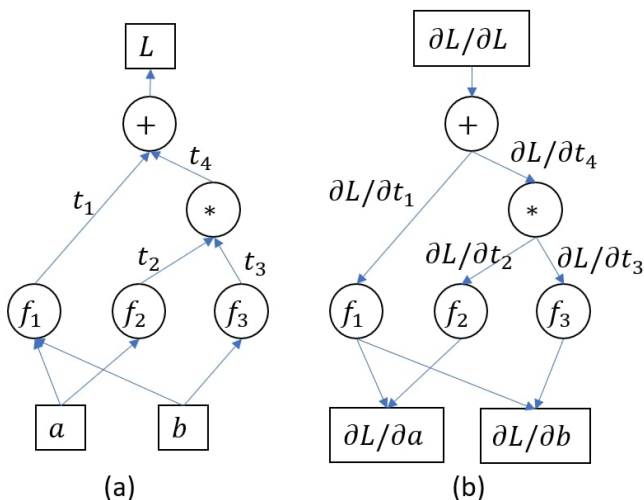


FIG. 1. The computational graph of calculation $L = f_1(a, b) + f_2(a)f_3(b)$. (a) The computational graph for the forward calculation. Note that t_1 to t_4 are the temporary variables in this case. (b) The backward gradient calculation from the computational graph. For backward calculation, it starts from $\partial L/\partial L = 1$ at the top and ends at the bottom. Merging arrows in this backward computational graph means that the gradients are accumulated.

tomatically without having to implement them explicitly. It works by constructing a computational graph and propagating the graph backward when calculating the gradient. The backward propagation of the computational graph follows the chain rule of differentiation. It uses a dynamical approach by default, which means that the computational graph is constructed when the forward calculation is performed.

For example, let's say we want to calculate a variable L from two other variables, a and b with $L = f_1(a, b) + f_2(a)f_3(b)$ for some functions f_1 , f_2 , and f_3 . If the calculation of L is performed from a and b , PyTorch will construct the graph that consists of some operations as shown in Figure 1(a). To calculate the gradient, e.g. $\partial L/\partial a$, the calculations are performed backward in the computational graph. It starts from $\partial L/\partial L$ and propagate through until it reaches the gradient that we want, as shown in Figure 1(b).

As it goes through an operation in the backward calculation, the backward calculation of that operation is performed. For example, if the operation is $t_4 = t_2t_3$ (i.e. the $*$ node in Figure 1), then the backward operation is $\partial L/\partial t_2 = \partial L/\partial t_4 t_3$ and $\partial L/\partial t_3 = \partial L/\partial t_4 t_2$. If all the operations in a calculation have the backward calculations defined, then one can back-propagate the graph to calculate the gradients.

D. xitorch

PyTorch provides the backward calculations for commonly used operations, such as multiplication, addition, matrix multiplication, etc. However, backward calculations for functionals (such as root finding) are not available in PyTorch. This is where xitorch [17] comes in. xitorch provides backward calculations of functionals, i.e. functions that require other functions as their inputs.

One example that is used in writing DQC is equilibrium finding, i.e. find \mathbf{x} such that

$$\mathbf{x} = \mathbf{f}(\mathbf{x}, \theta), \quad (7)$$

where \mathbf{f} is a function and θ is a parameter of that function. If \mathbf{x} is used to calculate a value L and $\partial L/\partial \mathbf{x}$ is available from another calculation, then the gradient of L with respect to the parameter θ is given by the adjoint method [24],

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \mathbf{x}} \left(\mathbf{I} - \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right)^{-1} \left(\frac{\partial \mathbf{f}}{\partial \theta} \right). \quad (8)$$

Using the adjoint method reduces the memory requirements in computing the gradient, compared to direct backpropagation of the equilibrium finder calculation as done in ref. 13, because the latter method needs to save temporary values calculated in the equilibrium finder iterations while the former requires no such temporary variables.

The gradient calculation above is provided by xitorch. Other than equilibrium finding, xitorch also provides gradient calculation of other functionals, such as root finding, optimization, and initial value problem solver.

III. IMPLEMENTATION

DQC is implemented mainly using PyTorch as the automatic differentiation package and xitorch for differentiating through functionals. Other than those 2 general purpose libraries, DQC also uses libxc [25] to include exchange-correlation functionals from literature and libcint [26] for Gaussian-basis integrals. Those libraries are wrapped with PyTorch to provide the automatic differentiation capability where the details can be found in ref. 14.

DQC implements restricted and unrestricted HF and Kohn-Sham DFT calculations without periodic boundary conditions. The energy can be minimized using either SCF iterations with Broyden's good method [27] for Fock matrix updates, or with direct minimization using gradient descent with momentum [28]; more elaborate SCF convergence accelerators will be implemented at a later stage of the project. All parameters are differentiable with respect to any other parameter present in the calculation, including nuclear coordinates, atomic numbers, electron occupation numbers, as well as the basis

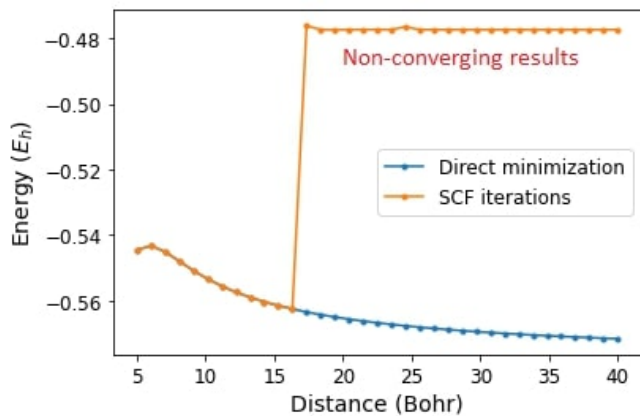


FIG. 2. Unrestricted PW92/cc-pVDZ [30, 31] energy for the hydrogen molecule cation H_2^+ as a function of the internuclear distance calculated via the SCF (using Broyden’s method [27]) and direct minimization approaches. Note that the SCF iterations do not converge for internuclear distances larger than 17 Bohr.

set exponents and contraction coefficients. The differentiability of these parameters allows for the exploration of new applications with DQC, a few of which are presented in what follows.

Although execution speed is not a top priority for DQC, the program shows good overall performance. Table I compares the running times of DQC with PySCF [29], an established quantum chemistry code. For small systems, we find that DQC is as efficient as PySCF. However, for moderate-sized molecules (e.g. C_4H_5N), DQC is about 4–6 times slower than PySCF. This slowdown can be attributed to DQC currently not taking advantage of the symmetry of the 2-electron integral tensor, which can reduce the tensor size by roughly a factor of 8 when the basis functions are real-valued. In contrast, DQC is only about 50% slower than PySCF for systems with density fitting. The difference is mainly caused by the higher number of algebraic operations in DQC than PySCF, because DQC has not been primarily optimized for speed. Speed optimization for DQC is subject to the future work.

IV. APPLICATIONS

A. Direct minimization

The automatic availability of gradients makes it easy to implement the direct minimization method in DQC. Direct minimization is known to be more robust than SCF iterations in finding a converged solution [19], and is particularly useful in difficult cases such as for molecules near their dissociation limits. We illustrate this in figure 2, where we show the PW92/cc-pVDZ [30, 31] total energy of H_2^+ as a function of the internuclear distance. The SCF method does not converge for internuclear dis-

TABLE II. Lowest eigenvalue of the HF/pc-1 orbital Hessian matrix. All of the calculations correspond to stationary points of the HF energy.

Molecules	Distance (Bohr)	Energy (E_h)	Lowest eigenvalue (E_h)
O ₂ (ground)	2.0	−149.54	-6×10^{-14}
O ₂ (excited)	2.0	−149.17	−0.73
BeH (ground)	2.5	−15.14	-4×10^{-14}
BeH (excited)	2.5	−15.01	−0.25
CH (ground)	2.0	−38.259	-5×10^{-8}
CH (excited)	2.0	−38.256	−0.07

tances greater than around 17 Bohr. In contrast, the direct minimization method continues to converge even for substantially larger distances in excess of 40 Bohr.

B. Checking SCF stability

One difficulty in SCF calculations is that the calculation can converge onto a saddle point, which corresponds to an excited state [19]. If the calculation has converged onto a local minimum, the Hessian of the energy with respect to orbital rotations must be positive semidefinite. Saddle point solutions, instead, correspond to one or more negative eigenvalues of the orbital Hessian. To maintain the orthonormality of the orbitals, the Hessian is calculated based on the \mathbf{Q} variable. Conveniently, we do not need to derive the expression for the Hessian matrix $\partial^2 \mathcal{E} / \partial \mathbf{Q}^2$ as it is automatically obtained by PyTorch.

Moreover, only the lowest eigenvalue needs to be calculated for the stability check, so the full Hessian matrix does not need to be constructed. We obtain the lowest eigenvalue using Davidson’s iterative algorithm [32], as implemented in xitorch. Note that the gradients of the lowest energy eigenvalue with respect to all other parameters in the calculation are automatically available, which may prove useful in further future applications.

We show some examples of SCF stability checks for HF/pc-1 [33] calculations on diatomic molecules in table II. As can be seen from the data, the lowest eigenvalues of the orbital Hessian are very close to zero for the ground state, while the excited states yield large negative eigenvalues. This illustrates that it is straightforward to check whether or not a state corresponds to a true minimum with DQC.

C. Molecular properties

A key advantage of writing quantum chemistry software with automatic differentiation is that calculations of molecular properties can be implemented efficiently: all we need to know is how a specific property relates to some derivative expression. For example, vibrational

TABLE III. Perturbative properties of H₂O from HF/cc-pVDZ calculations. The middle column presents the values calculated in DQC while the last column shows the CCCBDB values [34]. The IR intensities and Raman activities are presented at the frequency of 1800 cm⁻¹.

Properties	DQC	CCCBDB
IR intensities (km/mol)	80.69	80.70
Raman activities (Å ⁴ /amu)	4.79	4.79
Dipole (D)	-2.044	-2.044
Quadrupole _{xx} (DA)	-7.008	-7.008

modes and frequencies of a molecule can be written as

$$\mathbf{q}_{\text{vib}}, \omega_{\text{vib}} = \text{eig} \left(\frac{\partial^2 \mathcal{E}}{\partial \mathbf{X}^2} \right), \quad (9)$$

where \mathbf{X} is an $n \times 3$ matrix containing the nuclear coordinates of the n atoms, eig is the eigendecomposition, \mathbf{q}_{vib} is one of the vibrational modes, and ω_{vib} is its frequency. The expression shown in equation (9) is all that is needed to calculate the vibrational characteristics of the molecule; the explicit form of the derivative expression is not required.

Another useful example is the calculation of the electric dipole and quadrupole moments of a molecule. The electric dipole moment is given by

$$\boldsymbol{\mu} = -\frac{\partial \mathcal{E}}{\partial \mathbf{E}} + \sum_i Z_i \mathbf{x}_i, \quad (10)$$

while the electric quadrupole tensor is given by

$$\mathbf{M} = -2 \frac{\partial \mathcal{E}}{\partial \nabla \mathbf{E}} + \sum_i Z_i \mathbf{x}_i \mathbf{x}_i^T. \quad (11)$$

In both expressions, \mathcal{E} is the total energy of the molecule, \mathbf{E} is the electric field, Z_i is the atomic number of i -th atom, and \mathbf{x}_i are its coordinates.

Having these vibrational and electric multipole properties readily available makes obtaining the Raman vibrational spectrum straightforward. For example, the intensity of the infrared vibrational transition for a normal mode \mathbf{q} at a frequency ω is given by

$$I_{\text{IR}} = \sum_i \left(\sum_{jk} \frac{\partial \mu_i}{\partial X_{jk}} q_{jk} \right)^2, \quad (12)$$

where μ is the electric dipole moment and \mathbf{X} is the matrix of atomic coordinates.

Similarly, the Raman activity at the same frequency and normal mode is proportional to [35]

$$I_{\text{Raman}} = 5 [\text{tr}(\boldsymbol{\alpha})]^2 + 7\gamma; \quad (13)$$

$$\alpha_{ij} = \sum_{kl} \frac{\partial^2 \mu_i}{\partial E_j \partial X_{kl}} q_{kl}; \quad (14)$$

$$\gamma = \sum_{ij} (1 - \delta_{ij}) \left[\frac{1}{4} (\alpha_{ii} - \alpha_{jj})^2 + \frac{3}{2} \alpha_{ij}^2 \right], \quad (15)$$

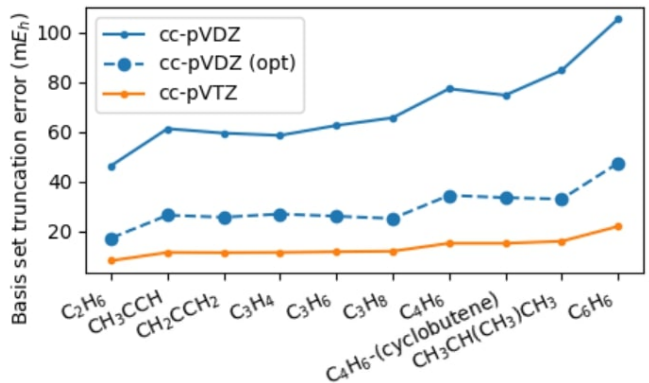


FIG. 3. Basis set truncation errors for the cc-pVDZ and cc-pVTZ basis sets for a range of hydrocarbons, compared with the truncation error for a reoptimized cc-pVDZ basis set. The reference energies are computed in the cc-pV5Z basis set. None of the molecules shown in the figure were used in the basis set optimization.

where δ_{ij} is the Kronecker delta.

The HF/cc-pVDZ perturbation properties of H₂O, found using the expressions above, are displayed in Table III. The values are in excellent agreement with data from the computational chemistry comparison and benchmark database (CCCBDB) [34], even though DQC does not have any explicit code to calculate the gradients required for these properties.

D. Basis set optimization

The differentiability of DQC with respect to the basis set parameters enables the optimization of system-specific basis sets. Here we show this capability by optimizing a basis set for hydrocarbons within Kohn–Sham DFT using the PW92 functional. We reoptimized the cc-pVDZ basis [31] for a training set of molecules consisting of CH (methylidyne), CH₃ (methyl radical), CH₄ (methane), C₂H₂ (acetylene), and C₂H₄ (ethylene). The accuracy of the reoptimized basis was then tested on a set of hydrocarbons that were not included in the training set. The results are shown in figure 3. The reoptimization of the cc-pVDZ basis set leads to a marked decrease of the total energies of all the molecules in the test set, as the cc-pVXZ basis set series is designed to capture correlation energies instead of polarization energies [31], and as hydrocarbons are chemically similar.

E. Alchemical perturbation

One of the differentiable quantities in DQC is the atomic number, allowing us to perform alchemical perturbation studies to predict the properties of molecules without actually needing to simulate them [36, 37]. As a simple example, we will show here that some properties

of diatomic molecules CO (atomic numbers 6 and 8) and BF (atomic numbers 5 and 9) can be estimated directly from the properties of N_2 (atomic number 7) and its alchemical perturbations. To do this, we parametrize the atomic number of the atoms in the diatomic molecule by a continuous variable λ , so that the atoms have atomic numbers $Z_l = 7 + \lambda$ and $Z_r = 7 - \lambda$. The molecules N_2 , CO, and BF thus correspond to $\lambda = 0$, $\lambda = 1$, and $\lambda = 2$, respectively.

The properties we aim to predict are the bond length s^* and the energy E^* at the equilibrium position, which can be mathematically expressed as

$$s^*(\lambda) = \arg \min_s E(s, \lambda), \quad (16)$$

$$E^*(\lambda) = E(s^*, \lambda). \quad (17)$$

Performing HF/pc-1 calculations, we evaluate the equilibrium distance and the energy at the equilibrium position in two ways. The first way is to optimize the geometry for various fixed values of λ , and calculate the above properties directly. The calculations were performed separately, but with the same basis (pc-1 nitrogen basis on all atoms) for different molecules, in analogy to the procedure used in refs 36 and 37. The second way is to estimate those properties using a Taylor series expansion to second order in λ :

$$s^*(\lambda) \approx s^*(0) + \lambda \frac{\partial s^*}{\partial \lambda}(0) + \frac{1}{2} \lambda^2 \frac{\partial^2 s^*}{\partial \lambda^2}(0) \quad (18)$$

$$E^*(\lambda) \approx E^*(0) + \lambda \frac{\partial E^*}{\partial \lambda}(0) + \frac{1}{2} \lambda^2 \frac{\partial^2 E^*}{\partial \lambda^2}(0). \quad (19)$$

As s^* and E^* are calculated at equilibrium, calculating the perturbation terms requires propagating the gradient through the optimization process. However, automatic differentiation makes the propagation trivial, as it is automatically handled by the optimization routine in xitorch.

The results obtained via these two approaches are compared in figure 4. As we can see from these results, the properties of CO and BF can be estimated accurately from alchemical perturbations of N_2 . The estimated equilibrium distances for CO and BF differ by -0.0004 and 0.024 Bohr, respectively, while the estimated equilibrium energies deviate by 33 and 587 m E_h , respectively. This shows that the equilibrium position of new molecules can be estimated well with the alchemical gradient calculated by DQC, without actually having to calculate those molecules.

V. DISCUSSION

Implementing quantum chemistry with automatic differentiation libraries is a promising way to accelerate simulation workflows and to enable novel applications. However, the implementation comes with several challenges. An overarching challenge is that the automatic differentiation library used here, PyTorch, is primarily designed

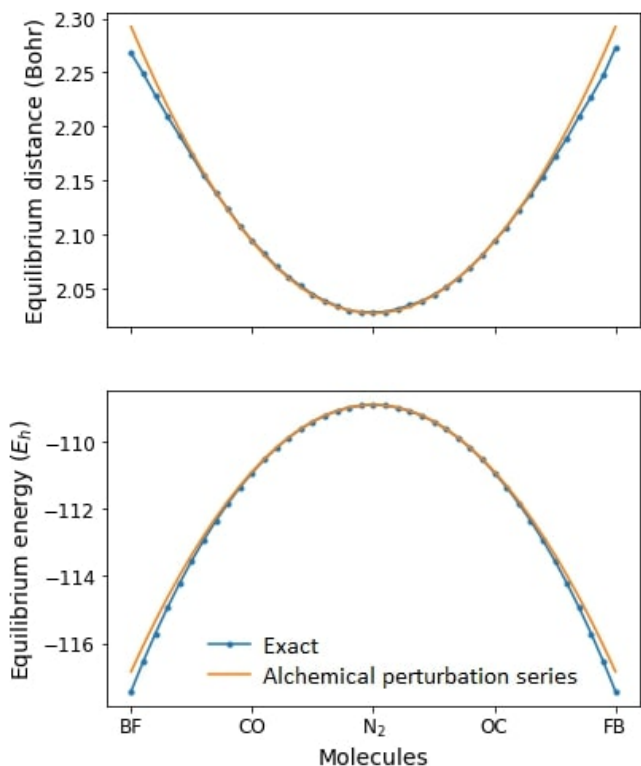


FIG. 4. Comparison of the properties of CO and BF obtained via an exact calculation and an estimation from the second-order gradients via alchemical perturbations, employing the nitrogen pc-1 basis on all atoms.

for deep learning rather than for scientific computing. As deep learning only focuses on low-order derivatives, accessing high-order gradients that are commonly required for scientific applications can be difficult.

Detecting numerical instabilities in high-order gradient calculations can also be demanding. Instabilities that produce NaN (not-a-number) in PyTorch are relatively straightforward to manage with its debugging feature since version 1.8, but other instabilities that do not produce a NaN can be challenging to detect.

Another challenge is debugging and profiling higher-order gradient calculations. As the gradient is generated automatically, it is hard to find the slowest part of the code, or the part that requires the most memory, because this information is not readily provided by the available profiling tools.

Besides higher-order gradient calculations, using automatic differentiation libraries also poses unique challenges in terms of code optimization. For example, quantum chemistry codes usually save the two-electron integrals on disk due to their large size and process them only in blocks small enough to fit easily into memory. This scheme can only be used in DQC if the gradients with respect to the nuclear positions and the basis are not required. To the authors' best knowledge, there is currently no obvious structure in PyTorch to allow gradient-

propagating operations to work with large tensors saved on disk.

Other automatic differentiation libraries developed for deep learning, such as Tensorflow [38] or JAX [39] can also be used to implement differentiable quantum chemistry. As those libraries have developed over the years, their features have become similar to each other. Therefore, differentiable quantum chemistry can also be written using those libraries. The choice depends on the preference of the developers; for instance, which library they are most familiar with.

VI. CONCLUSIONS

Implementing quantum chemical calculations using automatic differentiation liberates us from needing to derive analytical gradient expressions. With gradients automatically generated by the program, software developers can focus on designing better, more detailed computational models, and on applying them to problems at

hand. We have shown how automatic differentiation allows us to easily explore various applications in quantum chemistry, and are confident that further exploration of this approach will unveil new applications that have not been considered to date.

CODE AVAILABILITY

The Differentiable Quantum Chemistry (DQC) code can be found at <https://github.com/diffqc/dqc/>. The repository that contains the applications presented in this paper is located at <https://github.com/diffqc/dqc-apps/>.

ACKNOWLEDGMENTS

M.F.K. and S.M.V. acknowledge support from the UK EPSRC grant EP/P015794/1 and the Royal Society. S.M.V. is a Royal Society University Research Fellow. The authors declare no conflict of interest.

-
- [1] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.
 - [2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.
 - [3] Kristof T Schütt, Huziel E Sauceda, P-J Kindermans, Alexandre Tkatchenko, and K-R Müller. Schnet—a deep learning architecture for molecules and materials. *J. Chem. Phys.*, 148(24):241722, 2018.
 - [4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
 - [5] Hai-Jun Liao, Jin-Guo Liu, Lei Wang, and Tao Xiang. Differentiable programming tensor networks. *Physical Review X*, 9(3):031041, 2019.
 - [6] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *Conference on Robot Learning*, pages 317–335. PMLR, 2018.
 - [7] Samuel Schoenholz and Ekin Dogus Cubuk. Jax md: a framework for differentiable physics. *Advances in Neural Information Processing Systems*, 33, 2020.
 - [8] Teresa Tamayo-Mendoza, Christoph Kreisbeck, Roland Lindh, and Alán Aspuru-Guzik. Automatic differentiation in quantum chemistry with applications to fully variational hartree–fock. *ACS central science*, 4(5):559–566, 2018.
 - [9] Ulf Ekström, Lucas Visscher, Radovan Bast, Andreas J Thorvaldsen, and Kenneth Ruud. Arbitrary-order density functional response theory from automatic differentiation. *Journal of chemical theory and computation*, 6(7):1971–1980, 2010.
 - [10] Pierre Hohenberg and Walter Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136(3B):B864, 1964.
 - [11] Walter Kohn and Lu Jeu Sham. Self-consistent equations including exchange and correlation effects. *Phys. Rev.*, 140(4A):A1133, 1965.
 - [12] Adam S Abbott, Boyi Z Abbott, Justin M Turney, and Henry F Schaefer III. Arbitrary-order derivatives of quantum chemical methods via automatic differentiation. *The Journal of Physical Chemistry Letters*, 12(12):3232–3239, 2021.
 - [13] Li Li, Stephan Hoyer, Ryan Pederson, Ruoxi Sun, Ekin D Cubuk, Patrick Riley, and Kieron Burke. Kohn-Sham equations as regularizer: Building prior knowledge into machine-learned physics. *Phys. Rev. Lett.*, 126(3):036401, 2021.
 - [14] Muhammad F Kasim and Sam M Vinko. Learning the exchange-correlation functional from nature with fully differentiable density functional theory. *arXiv preprint arXiv:2102.04229*, 2021.
 - [15] Douglas Rayne Hartree. The wave mechanics of an atom with a non-coulomb central field. part ii. some results and discussion. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 24, pages 111–132. Cambridge University Press, 1928.
 - [16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 8026–8037. Curran Associates, Inc., 2019.

- [17] Muhammad F Kasim and Sam M Vinko. ξ -torch: differentiable scientific computing library. *arXiv preprint arXiv:2010.01921*, 2020.
- [18] Clemens Carel Johannes Roothaan. New developments in molecular orbital theory. *Rev. Mod. Phys.*, 23(2):69, 1951.
- [19] Susi Lehtola, Frank Blockhuys, and Christian Van Alsenoy. An overview of self-consistent field calculations within finite basis sets. *Molecules*, 25(5):1218, 2020.
- [20] Per-Olov Löwdin. Quantum theory of cohesive properties of solids. *Adv. Phys.*, 5(17):1–171, 1956.
- [21] Susi Lehtola. Curing basis set overcompleteness with pivoted Cholesky decompositions. *J. Chem. Phys.*, 151(24):241102, 2019.
- [22] Martin Head-Gordon and John A Pople. Optimization of wave function and geometry in the finite basis Hartree–Fock method. *J. Phys. Chem.*, 92(11):3063–3069, 1988.
- [23] Troy Van Voorhis and Martin Head-Gordon. A geometric approach to direct minimization. *Molecular Physics*, 100(11):1713–1721, 2002.
- [24] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.
- [25] Susi Lehtola, Conrad Steigemann, Micael JT Oliveira, and Miguel AL Marques. Recent developments in libxc—a comprehensive library of functionals for density functional theory. *SoftwareX*, 7:1–5, 2018.
- [26] Qiming Sun. Libcint: An efficient general integral library for gaussian basis functions. *Journal of computational chemistry*, 36(22):1664–1671, 2015.
- [27] Charles G Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation*, 19(92):577–593, 1965.
- [28] Barak A Pearlmutter. Gradient descent: Second-order momentum and saturating error. *Advances in neural information processing systems*, pages 887–894, 1991.
- [29] Qiming Sun, Xing Zhang, Samragni Banerjee, Peng Bao, Marc Barbry, Nick S Blunt, Nikolay A Bogdanov, George H Booth, Jia Chen, Zhi-Hao Cui, et al. Recent developments in the PySCF program package. *J. Chem. Phys.*, 153(2):024109, 2020.
- [30] John P Perdew and Yue Wang. Accurate and simple analytic representation of the electron-gas correlation energy. *Phys. Rev. B*, 45(23):13244, 1992.
- [31] Thom H Dunning Jr. Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen. *J. Chem. Phys.*, 90(2):1007–1023, 1989.
- [32] E. R. Davidson. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real symmetric matrices. *Journal of Computational Physics*, 17(1):87–94, 1975.
- [33] Frank Jensen. Polarization consistent basis sets: Principles. *J. Chem. Phys.*, 115:9113–9125, 2001.
- [34] Editor: R. D. Johnson III. NIST Computational Chemistry Comparison and Benchmark Database. NIST Standard Reference Database Number 101 (Release 21, August 2020), [Online]. Available: <https://dx.doi.org/10.18434/T47C7Z>. National Institute of Standards and Technology, Gaithersburg, MD., 2020.
- [35] Darragh P. O’Neill, Mihály Kállay, and Jürgen Gauss. Analytic evaluation of raman intensities in coupled-cluster theory. *Molecular Physics*, 105(19-22):2447–2453, 2007.
- [36] Robert Balawender, Michael Lesiuk, Frank De Proft, Christian Van Alsenoy, and Paul Geerlings. Exploring chemical space with alchemical derivatives: alchemical transformations of h through ar and their ions as a proof of concept. *Physical Chemistry Chemical Physics*, 21(43):23865–23879, 2019.
- [37] Guido Falk von Rudorff and O Anatole von Lilienfeld. Alchemical perturbation density functional theory. *Physical Review Research*, 2(2):023220, 2020.
- [38] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.
- [39] Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. *Systems for Machine Learning*, 2018.