The Journal of Chemical Physics

RESEARCH ARTICLE | MARCH 17 2023

PyQMC: An all-Python real-space quantum Monte Carlo module in PySCF

William A. Wheeler ; Shivesh Pathak ; Kevin G. Kleiner ; Shunyue Yuan ; João N. B. Rodrigues ; Cooper Lorsung ; Kittithat Krongchon ; Yueqing Chang ; Yiqing Zhou ; Brian Busemeyer; Kiel T. Williams; Alexander Muñoz ; Chun Yu Chow ; Lucas K. Wagner

J. Chem. Phys. 158, 114801 (2023)
https://doi.org/10.1063/5.0139024

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

PyQMC: An all-Python real-space quantum Monte Carlo module in PySCF

Cite as: J. Chem. Phys. 158, 114801 (2023); doi: 10.1063/5.0139024
Submitted: 16 December 2022 • Accepted: 23 February 2023 •
Published Online: 17 March 2023

William A. Wheeler,[1] Shivesh Pathak,[2] Kevin G. Kleiner,[3] Shunyue Yuan,[4] João N. B. Rodrigues,[5] Cooper Lorsung,[6] Kittithat Krongchon,[3] Yueqing Chang,[7] Yiqing Zhou,[8] Brian Busemeyer,[9] Kiel T. Williams,[10] Alexander Muñoz,[3] Chun Yu Chow,[3] and Lucas K. Wagner[3,a)]

AFFILIATIONS

[1]Department of Materials Science and Engineering, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA
[2]Center for Computing Research, Sandia National Laboratories, Albuquerque, New Mexico 87123, USA
[3]Department of Physics, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA
[4]Department of Applied Physics and Materials Science, California Institute of Technology, Pasadena, California 91125, USA
[5]Centro de Ciências Naturais e Humanas, Universidade Federal do ABC-UFABC, Santo André, São Paulo 09210-580, Brazil
[6]Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, USA
[7]Department of Physics and Astronomy, Rutgers University, Piscataway, New Jersey 08854, USA
[8]Laboratory of Atomic and Solid State Physics, Cornell University, Ithaca, New York 14853, USA
[9]Millenium Management, New York, New York 10022, USA
[10]Dynata, Plano, Texas 75024, USA

[a)]**Author to whom correspondence should be addressed:** lkwagner@illinois.edu

ABSTRACT

We describe a new open-source Python-based package for high accuracy correlated electron calculations using quantum Monte Carlo (QMC) in real space: PyQMC. PyQMC implements modern versions of QMC algorithms in an accessible format, enabling algorithmic development and easy implementation of complex workflows. Tight integration with the PySCF environment allows for a simple comparison between QMC calculations and other many-body wave function techniques, as well as access to high accuracy trial wave functions.

I. INTRODUCTION

Ab initio calculations play an integral role in advancing our knowledge of molecules and materials. They link materials properties to physical mechanisms in pristine systems, eliminating many difficult-to-control experimental factors. Without the need for experimental inputs, *ab initio* calculations and models also accelerate the search and design of new materials.[1,2] Strongly correlated materials, including unconventional superconductors,[3] 2D materials,[4,5] and defect systems,[6,7] require computational approaches with careful treatment of electron correlation.[8]

Calculations have an inherent trade-off between accuracy and computational cost: more accurate methods scale more steeply with the number of electrons, and exact calculations scale exponentially with system size. Quantum Monte Carlo (QMC) offers a good balance between accuracy and scalability, capable of treating systems with thousands of electrons.[9–12] The past few years have seen several advances in QMC methods: new wave functions using machine learning techniques,[13–18] new algorithms for optimizing excited states,[19–25] complex observables, such as energy density[26,27] and density matrices,[28] a new method to derive effective Hamiltonians from *ab initio* QMC,[29–31] and new time-stepping algorithms to reduce timestep error.[32,33]

Developing new tools and expanding the reach of QMC-level accuracy are necessary to address current problems in condensed matter physics but come with challenges. Achieving the highest performance can depend on subtle details of algorithm implementation,[33] and adding new methods can require significant changes

to algorithms. A bottleneck in this development process is the testing and implementation of new ideas in code. Several high-performance real-space QMC codes are under active development, including QMCPACK,[34] CASINO,[12] TurboRVB,[35] and CHAMP.[36] These real-space QMC software packages are written in low-level compiled languages, such as C++ and/or Fortran,[12,34–36] to achieve high performance suitable for large-scale calculations; however, these packages are bulky (many lines of code) and are challenging to modify.

To streamline the development and teaching of new ideas in quantum Monte Carlo, we have written PyQMC, an all-Python, flexible implementation of real-space QMC for molecules and materials. PyQMC is part of the PySCF ecosystem, a collection of libraries that achieve performance close to that of compiled languages while being implemented in the much more flexible Python language. In this manuscript, we will describe the implementation of PyQMC and note some of its advantages: integration with PySCF, fast development, modularity and compatibility with user-modified code, the flexibility of parallelization across diverse platforms (traditional desktop, cloud, and high performance computing), and a unified codebase for running on graphics processing units or central processing units.

II. QMC IMPLEMENTATION

There are many resources that offer thorough introductions to real-space QMC methods.[9–11,37–42] Here, we will describe our implementation of these methods in PyQMC.

A. Flexible wave functions

Wave functions are represented as Python objects in PyQMC. The standard implementation is the multi-Slater Jastrow (MSJ) wave function, having the form

$$\Psi(\mathbf{R}) = e^{J(\mathbf{R},\boldsymbol{\alpha})} \sum_k c_k D_k(\mathbf{R},\boldsymbol{\beta}), \tag{1}$$

where $\mathbf{R}$ represents the positions of all the electrons, and $\boldsymbol{\alpha}$ (Jastrow), $\mathbf{c}$ (determinant), and $\boldsymbol{\beta}$ (orbital) are variational parameters. Each determinant $D_k = \det\{\phi_i^k(\mathbf{r}_j)\}$ is constructed from a different set of single-particle orbitals $\{\phi_i\}$, where $\mathbf{r}_j$ is the position of electron j. The two-body Jastrow,

$$J(\mathbf{R};\boldsymbol{\alpha}) = \sum_{(i,j),I} u(r_{ij}, r_{iI}; \alpha), \tag{2}$$

is a function of all the electron–electron (r_{ij}) and electron–nucleus (r_{iI}) distances, where u is a function describing the cusp conditions and short-range correlation, defined in Ref. 44. These wave functions are compatible with both open and twisted boundary calculations.

The MSJ trial function allows for a compact representation of the wave function by using fewer determinants to represent static correlation and the Jastrow factor to represent the dynamical correlation.[45] Figure 1 compares the number of determinants needed with and without a Jastrow factor for a chain of six hydrogen atoms with a lattice spacing of 3.0 bohrs, in the strongly correlated regime. The variational Monte Carlo (VMC) calculation uses a two-body Jastrow with electron–electron and electron–ion pair correlation. Fixed-node diffusion Monte Carlo (DMC) can be interpreted as using the best possible Jastrow factor, shifting the wave function distribution without changing the nodal surface. Coupled cluster singles and doubles (CCSD) in the V5Z basis is near the complete basis set limit and is consistent with the DMC energy. Heat-bath configuration interaction (HCI) approaches the CCSD value as the determinant basis size increases. The pure determinant methods require two orders of magnitude more determinants than the VMC with a two-body Jastrow to converge.

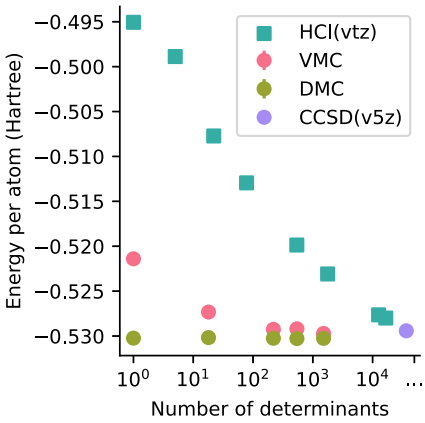


FIG. 1. Ground state energy of six-atom hydrogen chain at a spacing of 3 bohrs. Because the nodal error is small, DMC performs well with only a single determinant. VMC with a two-body Jastrow achieves a similar result with an order of magnitude fewer determinants than the pure multi-determinant methods. This figure is reproduced from Ref. 43.

In addition, we have implemented GeminalJastrow, the three-body Jastrow proposed by Sorella *et al.*[46] Any number of wave functions can be combined through the MultiplyWF and AddWF objects, enabling the mixing and matching of wave function forms. For efficiency, the Slater object includes a linear combination of determinants without the need for combining multiple wave function objects. As a subject of active research, we expect additional wave function forms to be added over time.

New wave functions are easily implemented in the PyQMC framework. Any object that conforms to the wave function interface can be used in all PyQMC methods. For example, other groups have implemented neural network trial functions[17] and used the algorithm outlined in Sec. II D 1 to optimize the wave function parameters. PyQMC's testing framework makes it possible to quickly check for bugs and ensure compatibility of new objects for seamless integration.

B. Expectation values of arbitrary operators

An arbitrary operator $\hat{O}$ is evaluated on wave functions Φ and Ψ as follows:

$$\frac{\langle\Phi|\hat{O}|\Psi\rangle}{\langle\Phi|\Psi\rangle} = \frac{\int d\mathbf{R}d\mathbf{R}'\Phi^*(\mathbf{R})\Psi(\mathbf{R}')O(\mathbf{R},\mathbf{R}')}{\int d\mathbf{R}\Phi^*(\mathbf{R})\Psi(\mathbf{R})}, \tag{3}$$

$$= \frac{\int d\mathbf{R}\Phi^*(\mathbf{R})\Psi(\mathbf{R}) \int d\mathbf{R}' \frac{\Psi(\mathbf{R}')}{\Psi(\mathbf{R})} O(\mathbf{R},\mathbf{R}')}{\int d\mathbf{R}\Phi^*(\mathbf{R})\Psi(\mathbf{R})} \tag{4}$$

$$= \frac{\int d\mathbf{R}\Phi^*(\mathbf{R})\Psi(\mathbf{R})\, O_L(\mathbf{R},\Psi)}{\int d\mathbf{R}\Phi^*(\mathbf{R})\Psi(\mathbf{R})}, \tag{5}$$

where the highlighted term is the local evaluation of the operator $\hat{O}$,

$$O_L(\mathbf{R},\Psi) = \int d\mathbf{R}' \frac{\Psi(\mathbf{R}')O(\mathbf{R},\mathbf{R}')}{\Psi(\mathbf{R})}. \tag{6}$$

In PyQMC, the integral over $\mathbf{R}$ is handled by the VMC algorithm, where $\Psi = \Phi = \Psi_T$, and in the case of DMC, $\Psi = \Psi_T$ is the trial function while $\Phi = \Phi_{FN}$ is the fixed-node projected wave function. We define an accumulator as an object that evaluates $\mathcal{O}_L(\mathbf{R},\Psi)$.

In this section, we summarize the accumulator objects implemented in PyQMC.

1. Gradient operators

For semilocal operators, such as gradients, the expression in Eq. (6) simplifies to

$$\mathcal{O}_L(\mathbf{R}) = \frac{[\hat{O}\Psi](\mathbf{R})}{\Psi(\mathbf{R})}. \tag{7}$$

In PyQMC, all wave function objects can compute $\frac{\nabla_r\Psi}{\Psi}$, $\frac{\nabla_r^2\Psi}{\Psi}$, and $\frac{\nabla_p\Psi}{\Psi}$, where ∇_r refers to the gradient with respect to a single electronic coordinate, and ∇_p refers to the gradient with respect to all variational parameters in the wave function.

2. Effective core potentials

PyQMC is compatible with semilocal effective core potentials (ECPs; nonlocal in the angular part, but local in the radial part). ECP evaluation is implemented as in QWalk[47] using the form described by Mitáš *et al.*[48] PyQMC automatically reads the ECPs from the PySCF mole or cell object.

The nonlocal operator takes the form of Eq. (6). The ECP operator $H^{\mathrm{ECP}}(\mathbf{R},\mathbf{R}') = \sum_{e,a} H^{\mathrm{ECP}}_{ea}(\mathbf{R},\mathbf{R}')$ is a sum of independent terms between electron e and atom a,

$$H^{\mathrm{ECP}}_{ea}(\mathbf{R},\mathbf{R}') = \delta(r_{ea} - r'_{ea}) \sum_l \frac{2l+1}{4\pi} v_l(r_{ea}) P_l(\cos\theta'), \tag{8}$$

where r_{ea} is the distance between positions of electron e and atom a, v_l is a radial pseudopotential for angular momentum channel l, P_l is a Legendre polynomial, and θ' is the angle between $\mathbf{r}_{ea}$ and $\mathbf{r}'_{ea}$. The angular integral for each (e,a) pair is evaluated using a randomly oriented quadrature rule,

$$\int d\mathbf{R}' H^{\mathrm{ECP}}_{ea}(\mathbf{R},\mathbf{R}') \frac{\Psi(\mathbf{R}')}{\Psi(\mathbf{R})} = \frac{4\pi}{N_\Omega} \sum_\Omega w_\Omega H^{\mathrm{ECP}}_{ea}(\mathbf{R},\mathbf{R}') \frac{\Psi(\mathbf{R}'_{ea\Omega})}{\Psi(\mathbf{R})},$$

where the auxiliary configurations $\mathbf{R}'_{ea\Omega}$ are generated from $\mathbf{R}$ by moving electron e about ion a by angles $\Omega = (\theta,\phi)$ of the quadrature grid and corresponding weights w_Ω. All the quadrature rules of octahedral and icosahedral symmetries listed by Mitáš *et al.*[48] are implemented in PyQMC.

3. Reduced density matrices

All one-particle observables can be calculated from the one-particle reduced density matrix (1-RDM), making it a useful quantity to characterize many-body wave functions alongside the energy. In PyQMC, the 1-RDM is represented in a basis of single-particle orbitals $\phi_i(\mathbf{r})$ as

$$\rho_{ij} = \langle\Psi|c_i^\dagger c_j|\Psi\rangle, \tag{9}$$

where $c_i^\dagger$ and c_i are creation and annihilation operators for orbital ϕ_i, respectively. Since the reduced density matrices are completely nonlocal, we perform an auxiliary random walk, sampling a conditional probability $P'(\mathbf{R}'|\mathbf{R})$ and evaluating

$$\mathcal{O}(\mathbf{R}) = \left\langle \frac{\Psi(\mathbf{R}')O(\mathbf{R},\mathbf{R}')}{\Psi(\mathbf{R})P'(\mathbf{R}'|\mathbf{R})} \right\rangle_{\mathbf{R}'\sim P'(\mathbf{R}'|\mathbf{R})}. \tag{10}$$

The 1-RDM is evaluated in QMC by averaging the quantity[28]

$$\rho_{ij} = \frac{1}{\sqrt{N_i N_j}} \left\langle \sum_{a=1}^{N} \frac{\Psi^*(\mathbf{R}'_a)}{\Psi^*(\mathbf{R})} \frac{\phi_i(\mathbf{r}'_a)\phi_j^*(\mathbf{r}_a)}{\rho_{\mathrm{aux}}(\mathbf{r}'_a)} \right\rangle_{\substack{\mathbf{R}\sim|\Psi|^2;\\ \mathbf{r}'_a\sim\rho_{\mathrm{aux}}}}, \tag{11}$$

where $\mathbf{R}'_a$ is generated from $\mathbf{R}$ by moving electron a, $\mathbf{r}_a \to \mathbf{r}'_a$, and $\rho_{\mathrm{aux}}(\mathbf{r}) = \sum_i |\phi_i(\mathbf{r})|^2$ is proportional to the one-particle distribution used to sample the auxiliary coordinate $\mathbf{r}'_a$. We use McMillan's method of using the same auxiliary coordinates $\mathbf{r}'_a$ for every electron a in the sum.[49] The normalization factors

$$N_i = \left\langle \frac{|\phi_i(\mathbf{r})|^2}{\rho_{\mathrm{aux}}(\mathbf{r})} \right\rangle_{\mathbf{r}\sim\rho_{\mathrm{aux}}} \tag{12}$$

are accumulated during the Monte Carlo run and are applied as a post-processing step using the function `normalize_obdm`.

The two-particle reduced density matrix (2-RDM)

$$\rho_{ijkl} = \langle\Psi|c_i^\dagger c_k^\dagger c_l c_j|\Psi\rangle \tag{13}$$

can be used to calculate all two-body observables and is analogous to the 1-RDM. Note that in some modules in PySCF and other quantum chemistry codes, $\langle\Psi|c_i^\dagger c_j c_k^\dagger c_l|\Psi\rangle$ is evaluated instead. It is relatively easy to translate between these two representations as

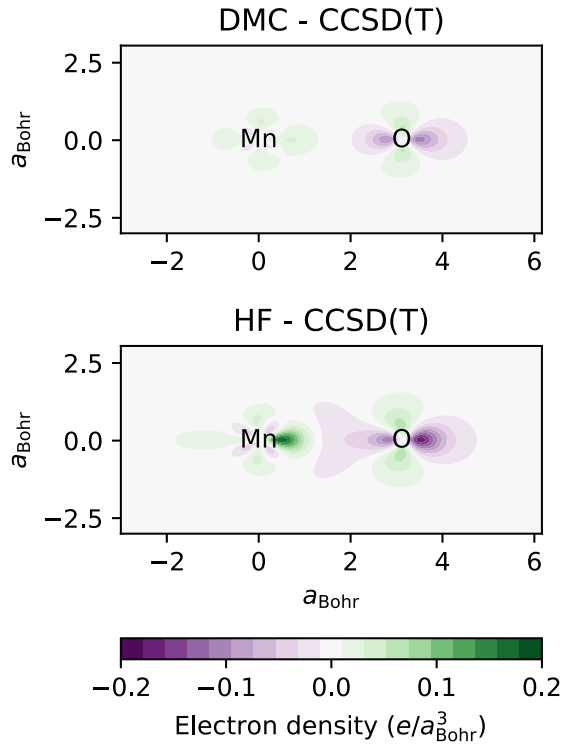
$$\langle\Psi|c_i^\dagger c_k^\dagger c_l c_j|\Psi\rangle = \langle\Psi|c_i^\dagger c_j c_k^\dagger c_l|\Psi\rangle - \delta_{jk}\langle\Psi|c_i^\dagger c_l|\Psi\rangle, \tag{14}$$

which is done by the PySCF function `reorder_rdm`. The 2-RDM is evaluated in QMC as

$$\rho_{ijkl} = \left\langle \sum_{a<b} \frac{\Psi(\mathbf{R}'_{ab})}{\Psi(\mathbf{R})} \frac{\phi_j^*(\mathbf{r}'_a)\phi_l^*(\mathbf{r}'_b)\phi_i(\mathbf{r}_a)\phi_k(\mathbf{r}_b)}{\rho_{\mathrm{aux}}(\mathbf{r}'_a)\rho_{\mathrm{aux}}(\mathbf{r}'_b)} \right\rangle_{\substack{\mathbf{R}\sim|\Psi|^2;\\ \mathbf{r}'_a\sim\rho_{\mathrm{aux}}}}, \tag{15}$$

PyQMC's implementation can evaluate the RDMs on an arbitrary basis.

PySCF routines can be applied directly to the 1-RDM computed in DMC to compute and plot density or other one-body quantities (Fig. 2). Using PySCF's built-in `cubegen.density` function removes the need to write a new script for plotting.




```python
import numpy as np
import pyqmc.api as pyq
import pyscf.tools

mol, mf = pyq.recover_pyscf("MnO.chk")
rdm1_dmc = pyq.read_mc_output(
    "MnO_DMC.hdf5",
    warmup=100,
    reblock=20,
)
rdm1_ccsd_t = np.load("MnO_CCSD(T).npy")
ao_rdm1_dmc = np.einsum(
    "pi,ij,qj->pq",
    mf.mo_coeff,
    rdm1_dmc - rdm1_ccsd_t,
    mf.mo_coeff.conj()
)
dens_dmc = pyscf.tools.cubegen.density(
    mol, "MnO_dmc.cube", ao_rdm1_dmc
)
```

FIG. 2. The integration between PySCF and PyQMC makes it straightforward to compare properties of wave functions between different high level methods. (a) Electron density of the MnO molecule at bond length 1.6477 Å for Hartree–Fock and diffusion Monte Carlo referenced to CCSD(T) calculations. (b) The code used to compute the DMC densities. For the entire code used to generate the plots, see the supplementary material.

Computing RDMs on the same basis allows for seamless comparison between methods, i.e., by simply subtracting the matrices. Different methods are commonly compared by their energies, a single number. One- and two-particle density matrices capture more of the state and offer a better comparison of properties; methods that result in the same energy may still produce states with different densities. QMC computations of RDMs on a basis have an added advantage in that the statistical noise is much smaller compared to computing on a grid, resulting in smoother density plots. The difference in densities between Hartree–Fock, DMC, and coupled cluster singles and doubles with perturbative triples [CCSD(T)] is shown in Fig. 2.

C. Bulk systems

Infinite solids are approximated by finite simulation cells with twisted boundary conditions (TBCs),

$$\Psi(\mathbf{r}_1,\ldots,\mathbf{r}_i+\mathbf{L},\ldots,\mathbf{r}_N) = e^{i\mathbf{k}\cdot\mathbf{L}}\Psi(\mathbf{r}_1,\ldots,\mathbf{r}_i,\ldots,\mathbf{r}_N), \tag{16}$$

where $\mathbf{k}$ is the twist; thus, the basis functions are eigenstates of a translation operator. The one-particle part of the Hamiltonian commutes with the translation of a single electron and, thus, can be diagonalized using basis functions of definite twist. The total energy per cell is obtained by averaging over all twists in the Brillouin zone.[50] However, the Coulomb operator does not commute with the translation operator of a single electron and, thus, causes the energy eigenstates to, in general, be superpositions of twists.

In PyQMC, practical calculations are performed using a *supercell* approximation, in which a simulation cell larger than the primitive cell is chosen, and the Coulomb operator is truncated to remove matrix elements between different twists. This truncation can be partially corrected using the structure factor,[51] with an error proportional to $\frac{1}{N}$, where N is the number of electrons in three dimensions. Note that this correction should be performed after twist averaging above.

PyQMC contains several features to facilitate extrapolation to infinite system size. First, a PySCF mean-field calculation is performed on the primitive cell. The k-points used in the mean-field calculation determine which twists are available for a given supercell $\mathbf{S}$. The available twists are obtained in PyQMC using the function `available_twist(cell, mf, and S)`, where `cell` and `mf` are PySCF cell and mean-field objects. The code then automatically generates the appropriate supercell objects from the primitive cell mean-field object. By averaging over twist, one can remove the kinetic energy finite size correction.[51] The small-k limit of the structure factor gives the approximate Coulomb finite size correction.[51] The structure factor is available as an accumulator in PyQMC.

D. Methods

1. Variational Monte Carlo (VMC)

The trial functions in Sec. II A can contain hundreds or thousands of parameters. To approximate the ground state, the parameters of the trial function are variationally optimized by minimizing the VMC energy,

$$E[\Psi] = \langle\Psi|\hat{H}|\Psi\rangle = \left\langle \frac{\hat{H}\Psi(\mathbf{R})}{\Psi(\mathbf{R})} \right\rangle_{R\sim|\Psi|^2}. \tag{17}$$

The optimization algorithm is shown in Fig. 3. The gradient of $E[\Psi]$ is used to determine the updates to the parameters $\mathbf{p}$ during optimization. The gradient estimator $\frac{\hat{H}\Psi}{\Psi}\frac{\partial_p\Psi}{\Psi}$ has infinite variance near the nodes of Ψ, which is removed by including the regularization factor of Ref. 52. Next, the para-

1. Generate walkers $\mathbf{R}$
2. Compute regularization factor[52] $f(\mathbf{R})$
   (a) $d(\mathbf{R}) \leftarrow \frac{1}{r_{\text{cutoff}}} \frac{\Psi(\mathbf{R})}{\sqrt{\sum_e |\nabla_e \Psi(\mathbf{R})|^2}}$
   (b) $f(\mathbf{R}) \leftarrow 9d(\mathbf{R})^2 - 15d(\mathbf{R})^4 + 7d(\mathbf{R})^6$
3. Stochastic reconfiguration[53, 54]
   (a) $G^i_\Psi(\mathbf{R}) \leftarrow \frac{\partial_{p_i}\Psi}{\Psi}\big|_{\mathbf{R}} f(\mathbf{R})$
   (b) $E_L(\mathbf{R}) \leftarrow \frac{\hat{H}\Psi}{\Psi}\big|_{\mathbf{R}}$
   (c) $G^i_E \leftarrow \langle E_L(\mathbf{R}) G^i_\Psi(\mathbf{R})\rangle_{\mathbf{R}} - \langle E_L(\mathbf{R})\rangle_{\mathbf{R}} \langle G^i_\Psi(\mathbf{R})\rangle_{\mathbf{R}}$
   (d) $S_{ij} \leftarrow \left\langle \frac{\partial_{p_i}\Psi}{\Psi}\big|_{\mathbf{R}} G^j_\Psi(\mathbf{R}) \right\rangle_{\mathbf{R}} - \langle G^i_\Psi(\mathbf{R})\rangle_{\mathbf{R}} \left\langle G^j_\Psi(\mathbf{R})\right\rangle_{\mathbf{R}}$
   (e) $u_i \leftarrow \sum_j (\mathbf{S})^{-1}_{ij} G^j_E$ (regularized gradient)
4. Line minimization using correlated sampling
   (a) Select walkers $\mathbf{R}$
   (b) $\mathbf{p} \leftarrow$ parameters of Ψ
   (c) **for** x in [-1, 0, 1, 2, 3], **do**
      i. $\Psi_x \leftarrow$ replace $\mathbf{p}$ with $\mathbf{p} + x\mathbf{u}$
      ii. $E(x) = \left\langle \frac{\hat{H}\Psi_x}{\Psi_x} \left|\frac{\Psi_x}{\Psi_0}\right|^2 \right\rangle_{\mathbf{R}}$
   (d) $E_{\text{fit}} \leftarrow$ fit $E(x)$ to cubic function
   (e) $x_{\min} \leftarrow \arg\min_x E_{\text{fit}}(x)$
   (f) $\Psi \leftarrow \Psi_{x_{\min}}$

FIG. 3. Pseudo-code for the wave function optimization routine in PyQMC. The three main parts of each step of the optimization algorithm: variance regularization factor, stochastic reconfiguration, and line minimization.

meter update direction is determined from $\frac{\partial E}{\partial p}$ using the stochastic reconfiguration technique of Casula and Sorella.[53,54] Finally, the magnitude is determined by the minimum energy along the update direction. The parameters corresponding to the minimum are determined by a polynomial fit of correlated samples of the energy along the line. The parameters are updated, and the process is repeated to convergence.

For multi-Slater–Jastrow functions [Eq. (1)], PyQMC supports optimization of α (Jastrow), c (determinant), and β (orbital) parameters.

2. VMC for excited states

A standard approach to computing excited states is to hold orbital coefficients fixed from an excited mean-field determinant.[55] To optimize excited-state orbitals, additional measures are required to keep them from reverting to the orbitals of lower-energy states. Methods, such as the state-averaged complete active space self-consistent field (CASSCF) method[56,57] or other state averaged methods,[20,24,58–60] allow orbital shapes to vary but require the orbitals to be the same for all energy eigenstates. This requirement makes it easier to enforce the orthogonality of eigenstates but severely limits the expressiveness of the wave functions.

In PyQMC's implementation, excited states are kept orthogonal to lower-energy states through an overlap penalty introduced in Ref. 25, allowing orbital coefficients to be optimized for each state independently. The objective function for the optimization is given by

$$O[\Psi] = \langle\Psi|\hat{H}|\Psi\rangle + \sum_{i=1}^{n-1} \lambda_i |\langle\Psi_i|\Psi\rangle|^2 \tag{18}$$

$$= \left\langle \frac{\hat{H}\Psi(\mathbf{R})}{\Psi(\mathbf{R})} \frac{|\Psi(\mathbf{R})|^2}{\rho} \right\rangle + \sum_{i=1}^{n-1} \lambda_i \left| \frac{N_{in}}{\sqrt{N_{ii}N_{nn}}} \right|^2, \tag{19}$$

where

$$N_{ij} = \left\langle \frac{\Psi^*_i(\mathbf{R})\Psi_j(\mathbf{R})}{\rho(\mathbf{R})} \right\rangle, \tag{20}$$

is the wave function overlap matrix and $\mathbf{R}$ is sampled from the distribution $\rho(\mathbf{R})$. Typically, $\rho(\mathbf{R}) \propto \sum_i |\Psi_i(\mathbf{R})|^2$.

To demonstrate the importance of orbital optimization for excited states, we show optimizations of the ground and first excited states of the CO molecule (Fig. 4) using a 400-determinant multi-Slater–Jastrow ansatz, with and without optimizing orbitals. The energy is shown at each iteration over the course of both optimizations. Fixed-orbital wave functions yield an excitation energy of 9.43(5) eV, compared with 6.68(5) eV from optimized-orbital wave functions of the same form. Compared with the experimentally determined vertical excitation energy 4.76 eV,[61] optimizing orbitals results in a 60% improvement at the VMC level.

3. Diffusion Monte Carlo

Diffusion Monte Carlo (DMC) is implemented in PyQMC using importance sampling and the fixed-node approximation. Samples are drawn from the mixed distribution $f(\mathbf{R}) = |\Psi^*(\mathbf{R})\Phi_0(\mathbf{R})|$, where Φ_0 is the ground state, by stochastically applying a projection operator $\hat{\mathcal{P}}_\tau$ to a trial function Ψ,

$$\Psi^*(\mathbf{R})\Phi_0(\mathbf{R}) = \lim_{N\to\infty} \langle\Psi|\mathbf{R}\rangle\langle\mathbf{R}|\hat{\mathcal{P}}^N_\tau|\Psi\rangle. \tag{21}$$

The time step τ is a parameter that must be extrapolated to $\tau \to 0$. Positions and weights $(\mathbf{R}_i, w_i)$ are generated by the projection $\hat{\mathcal{P}}_\tau$ at each Monte Carlo step. The fixed-node approximation is used for real wave functions, rejecting moves $\mathbf{R} \to \mathbf{R}'$ that change the sign of the trial function Ψ. For complex wave functions, the fixed-phase approximation is used.[62] Because the gradient of the phase enters into the potential, no rejection based on the sign is required.

Sampling the mixed distribution results in mixed-estimator averages,

$$\langle\hat{O}\rangle = \frac{\langle\Psi|\hat{O}|\Phi_0\rangle}{\langle\Psi|\Phi_0\rangle}, \tag{22}$$

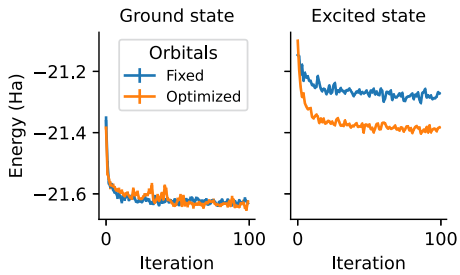


FIG. 4. Optimization of a minimal multi-Slater–Jastrow wave function for the ground and first excited states of a CO molecule, with and without optimizing the orbitals. Better variational estimates are achieved for the excited state by optimizing the orbitals.

which are computed similarly to Eq. (3) as averages over walkers with additional weights w_i,

$$\left\langle w_i \frac{\int d\mathbf{R}'\, O(\mathbf{R}_i, \mathbf{R}')\Psi^*(\mathbf{R}')}{\Psi^*(\mathbf{R}_i)} \right\rangle_{\mathbf{R}_i \sim f(\mathbf{R})}. \tag{23}$$

Branching is performed every few steps to keep weights balanced, replicating some walkers and removing others depending on their weights. In PyQMC, the branching is implemented by the stochastic comb method:[63–65] walkers are resampled with probability proportional to their weights, the total weight $\sum_j w_j$ is saved, and the new weights are subsequently set equal to one. The expected contribution from each walker is correct on average, and the resulting population bias is small. This approach has the advantage of keeping the number of walkers fixed, which simplifies efficient parallelization on a fixed number of processors.

PyQMC employs two strategies proposed by Anderson and Umrigar[33] to reduce time-step errors: modified weight updates and modified T-moves[66] for nonlocal ECPs.

III. DIVERSE WORKFLOW SUPPORT

A. Integration with PySCF

In many QMC codes, converters from other packages make up a large portion of the programming effort. PyQMC uses PySCF objects directly to initialize calculations, eliminating the need for converters. Mole and Cell objects define the Hamiltonian, including geometry, number of electrons, basis set, and pseudopotentials. The use of PySCF's `eval_gto()` function to evaluate orbitals guarantees compatibility with any basis set supported by PySCF. QMC trial wave function determinants are generated from SCF objects, and there is some compatibility with multireference methods, such as complete active space (CAS), CASSCF, and full CI without requiring conversion steps.

```python
from pyscf import gto, scf
import pyqmc.api as pyq

mol = gto.M(
    atom=f"Mn 0. 0. 0.; O 0. 0. 1.6477",
    basis="ccecp-ccpvtz",
    ecp="ccecp",
    spin=5,
)
mf = scf.UHF(mol)
mf.run()

# QMC
configs = pyq.initial_guess(mol, nconfig)
wf, to_opt = pyq.generate_wf(mol, mf)
pgrad_acc = pyq.gradient_generator(mol, wf,
    to_opt)
wf, optimization_data = pyq.line_minimization
    (wf, configs, pgrad_acc)
configs, dmc_data = pyq.rundmc(wf, configs)
```

FIG. 5. Single script execution of a QMC calculation from atomic positions to QMC result. PySCF objects are used directly in PyQMC functions. No writing intermediate results to disk is required.

Tight coupling to PySCF enables easy use of analysis routines. A common example is the calculation and plotting of density differences discussed in Sec. II B 3 and shown in Fig. 2.

PyQMC allows for file-free computation—executing a full calculation from atomic structure to QMC result without saving any intermediate results (Fig. 5). Having all objects and data in the workspace streamlines the prototyping of new algorithms and workflows.

B. Monkey patching

PyQMC allows users to add modifications to a calculation locally without changing the package directory, a practice known as "monkey patching." Although modifying the package directory is certainly possible, it poses a barrier to users in our experience. With PyQMC's all-Python, modular structure, built-in routines are compatible with objects defined outside the package directory, such as customized wave function and accumulator objects for VMC and DMC; built-in objects can be used in externally defined customized methods as well. Figure 6 shows code outside of the package defining an accumulator object that is used directly in PyQMC's VMC and DMC routines, in this case, to compute the dipole moment of

```python
import numpy as np
import pyqmc.api as pyq


class DipoleAccumulator:
    def __init__(self):
        pass

    def __call__(self, configs, wf):
        return {"electric_dipole": configs.
    configs.sum(axis=1)}

    def avg(self, configs, wf):
        avg = {}
        data = self(configs, wf)
        for k, it in data.items():
            avg[k] = np.mean(it, axis=0)
        return avg

    def shapes(self):
        return {"electric_dipole": (3,)}

    def keys(self):
        return self.shapes().keys()


accumulators = {"extra_accumulators": dict(
    dipole=DipoleAccumulator())}
pyq.VMC(
    "MnO_scf.hdf5",
    "MnO_vmc_dipole.hdf5",
    load_parameters="MnO_opt.hdf5",
    accumulators=accumulators,
)
```

FIG. 6. User code can be injected into PyQMC's QMC routines. In this example, we defined a class on the fly to compute the molecular dipole moment within the script that performs the calculation.

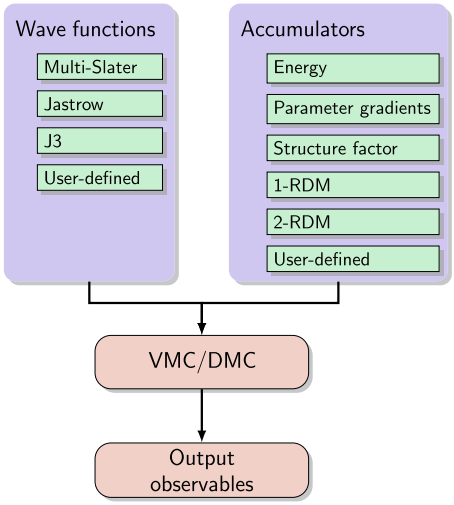


FIG. 7. User-defined code can be mixed and matched in multiple ways. Externally defined accumulators are input directly into built-in VMC and DMC routines, as in Fig. 6. Similarly, externally defined wave functions are simultaneously input directly into VMC and DMC.

a molecule. Using custom accumulator objects is depicted in the flowchart in Fig. 7.

As an example of the benefits of this platform, we contrast the implementation of a new VMC algorithm between Python and C++ (e.g., for sampling the sum of two wave functions in excited state optimizations). In C++, the new algorithm would require adding a file into the package, adding the file into the make system, and recompiling the distribution. In Python, a customized VMC is written, tested, and run at scale without the user modifying the distributed package at all, as depicted in the flowchart in Fig. 7. It is completely portable; the new algorithm file(s) can be shared, and it will work for another user or machine. Developing new QMC methods and algorithms is often iterative, and by requiring fewer steps, this Python implementation greatly reduces friction for users and developers to explore new ideas.

IV. ACCELERATION STRATEGIES

PyQMC supports two acceleration strategies: parallel execution, and the use of graphical processing units (GPUs). The strategies work simultaneously: quantum Monte Carlo calculations can use multiple GPUs across multiple computational resources. It is possible to parallelize on heterogeneous resources, in which some calculations are performed on central processing units (CPUs) and some on GPUs.

A. Parallelization

PyQMC makes use of Python's standard library `futures` objects for parallelization. For compatibility with PyQMC, a futures object need only implement the `submit` function, which distributes work onto a remote process or server. By using `futures` objects, PyQMC can transparently take advantage of many parallelization strategies. The Python standard library `concurrent.futures` provides on-node process-based parallelization. Other packages can be installed and used with the code transparently; for example, `mpi4py`[67–70] provides futures over the high performance computing standard Message Passing Interface.[71] Similarly, `Dask`[72] provides a futures-based interface using pilot processes that are very flexible, allowing for remote execution on cloud-based resources. Any interface for submitting tasks via futures objects, including custom user code, can be used for running PyQMC in parallel.

PyQMC implements the well-known MapReduce paradigm for parallel execution. A main process sends wave function objects to workers, which perform Monte Carlo moves and accumulate data. The workers then return the data to the main process. Because the futures' interface is used, the data can be sent using any backend that supports that interface. To keep most of the computation in NumPy functions (and thus executed in C/Fortran), it is important that each process perform the calculation with a sufficient number of walkers, which depends on the system size and can vary from just a few walkers for a large system to 500 for a very small system.

Quantum Monte Carlo methods are often termed "embarrassingly parallel," meaning that the computational time decreases almost linearly as the number of processors increases. Figure 8 shows the number of Monte Carlo steps executed per second as a function of the number of nodes used for a VMC calculation on a coronene molecule. The parallel efficiency on 64 Summit nodes (2688 cores) is above 99.9%. This scaling is representative of what one should expect in optimization, DMC, and excited state calculations (i.e., all types of calculations). Our flexible parallel implementation, thus, does not seem to have any disadvantage over more standard approaches using Message Passing Interface (MPI).

B. Graphical processing unit acceleration

PyQMC runs on CPUs and GPUs using the same code paths. GPU capability is implemented using the `CuPy` library,[73] which is used as a drop-in replacement for NumPy. Currently, wave function evaluation and Ewald summation, which are computationally intensive, run on GPU when available and return arrays on CPU. VMC,

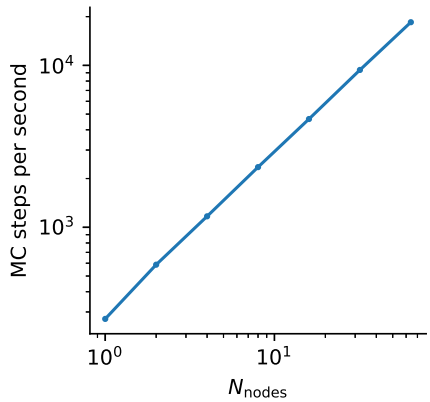


FIG. 8. Parallel scaling of VMC on a coronene molecule, with the number of walkers scaled proportionally to the number of cores. Calculations were run on Summit and parallelized with `Dask`.[72]

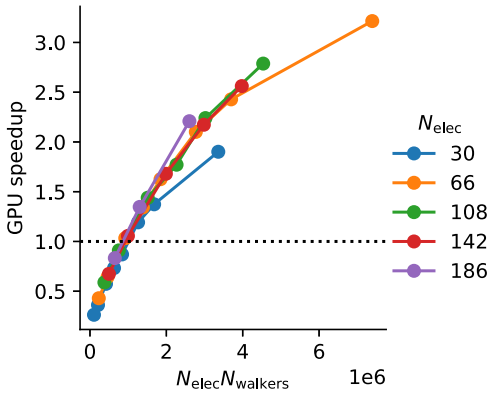


FIG. 9. GPU speedup (t_{CPU}/t_{GPU}) vs the product of the number of electrons and the number of walkers (amount of work) for a sequence of hydrocarbon molecules: benzene (30 electrons), anthracene (66 electrons), coronene (108 electrons), ovalene (142 electrons), and hexabenzocoronene (186 electrons). Comparisons are run on a single Summit node using 37 CPUs in both cases and six GPUs for the GPU case. The calculations were parallelized using Dask. For a large enough problem size, the GPU speedup depends only on the amount of work available.

DMC, and other algorithmic-level functions are coded entirely on the CPU; implementing new algorithms does not require any extra interfacing to make use of GPU resources.

Figure 9 shows the GPU speedup (ratio of CPU time to GPU time) vs $N_{walker}N_{elec}$ for a sequence of hydrocarbon molecules: benzene (30 electrons), anthracene (66 electrons), coronene (108 electrons), ovalene (142 electrons), and hexabenzocoronene (186 electrons). The calculations used correlation-consistent effective core potentials and corresponding VDZ basis sets for both H and C atoms from pseudopotentiallibrary.org.[74,75] Each calculation was carried out on a single node of the Summit supercomputer at Oak Ridge National Laboratory and parallelized using Dask.[72] For sufficiently large numbers of electrons (about 60–100), the speedup collapses onto a single line, which only depends on $N_{walker}N_{elec}$, approximately the amount of work given to the GPU.

We believe that there could be improvements to the GPU performance of the code by porting more of the code from the CPU to the GPU. In particular, `PyQMC` uses `PySCF`'s functions to evaluate the atomic orbitals on the CPU. For the molecules shown in Fig. 9, the atomic orbital evaluation takes up 15%–20% of the time, meaning that the GPU speedup in these tests is limited to a maximum of five or six, even if it performed the work in zero time with zero latency. In the future, we are, thus, targeting this bottleneck to achieve better GPU speedups.

V. CONCLUSION

`PyQMC` is a production-level, feature-complete, and state-of-the-art QMC implementation linked with `PySCF`. Because `PyQMC` is implemented entirely in Python, it is extremely flexible and modular. Similarly to `PySCF` for standard quantum chemistry methods, `PyQMC` is aimed at both production level calculations and the development of new methods. Just within our group and others, these features have already led to new algorithmic developments.[17,25,43,52,76] `PyQMC` is licensed under the MIT license[77–79] and is, thus, freely available to download and modify. Other groups are free to build on the base implementations laid out here.

Python's high level of abstraction greatly reduces the human time required to customize implementations and develop new ideas. The library ecosystem is well-developed, including libraries for scientific computing (`NumPy`[80] and `SciPy`[81]), data I/O (`h5py`[82]), parallelization (concurrent, `MPI for Python`,[70] and `Dask`[72]), and GPU execution (`CuPy`[73]). `PyQMC` is written in such a way that almost all computationally intensive tasks are actually executed in compiled C or Fortran code provided by one of those libraries so that the performance is competitive with packages implemented completely in compiled languages while code can be written at a high level.

SUPPLEMENTARY MATERIAL

The supplementary material contains all the code used to generate the MnO molecule density data shown in Fig. 2.

ACKNOWLEDGMENTS

We thank Scott Jensen for helping to read the manuscript. Support of W.A.W. and L.K.W. from the U.S. National Science Foundation via Award No. 1931258 is acknowledged for the development and integration of `PyQMC` into `PySCF`. Y.C. was supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, Computational Materials Sciences Program, under Award No. DE-SC0020177. Implementing GPU compatibility used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. Additional testing of GPU compatibility used HPC resources of the SDumont supercomputer at the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil). This work made use of the Illinois Campus Cluster, a computing resource that is operated by the Illinois Campus Cluster Program (ICCP) in conjunction with the National Center for Supercomputing Applications (NCSA), which is supported by funds from the University of Illinois at Urbana-Champaign.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

William A. Wheeler: Software (lead); Writing – original draft (lead). **Shivesh Pathak**: Software (supporting). **Kevin G. Kleiner**: Software (supporting); Visualization (supporting). **Shunyue Yuan**: Software (supporting). **João N. B. Rodrigues**: Software (supporting); Writing – review & editing (supporting). **Cooper Lorsung**: Software (supporting). **Kittithat Krongchon**: Software (supporting). **Yueqing Chang**: Software (supporting); Writing – review &

editing (supporting). **Yiqing Zhou**: Software (supporting). **Brian Busemeyer**: Methodology (supporting); Software (supporting). **Kiel T. Williams**: Software (supporting). **Alexander Muñoz**: Software (supporting); Writing – review & editing (supporting). **Chun Yu Chow**: Software (supporting). **Lucas K. Wagner**: Project administration (equal).

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request. The latest version of the `PyQMC` source code is available at https://github.com/WagnerGroup/pyqmc. `PyQMC` can also be installed from the Python Package Index (PyPI) via `pip install pyqmc` or `pip install pyscf[pyqmc]`.

REFERENCES

[1] S. Lebègue, T. Björkman, M. Klintenberg, R. M. Nieminen, and O. Eriksson, "Two-dimensional materials from data filtering and *ab initio* calculations," Phys. Rev. X **3**, 031002 (2013).

[2] S. Curtarolo, G. L. W. Hart, M. B. Nardelli, N. Mingo, S. Sanvito, and O. Levy, "The high-throughput highway to computational materials design," Nat. Mater. **12**, 191–201 (2013).

[3] J.-B. Morée, M. Hirayama, M. T. Schmid, Y. Yamaji, and M. Imada, "*Ab initio* low-energy effective Hamiltonians for high-temperature superconducting cuprates $Bi_2Sr_2CuO_6$, $Bi_2Sr_2CaCu_2O_8$, $HgBa_2CuO_4$ and $CaCuO_2$," Phys. Rev. B **106**, 235150 (2022).

[4] K. Choudhary, I. Kalish, R. Beams, and F. Tavazza, "High-throughput identification and characterization of two-dimensional materials using density functional theory," Sci. Rep. **7**, 5179 (2017).

[5] N. P. Wilson, W. Yao, J. Shan, and X. Xu, "Excitons and emergent quantum phenomena in stacked 2D semiconductors," Nature **599**, 383–392 (2021).

[6] Á. Gali, "*Ab initio* theory of the nitrogen-vacancy center in diamond," Nanophotonics **8**, 1907–1943 (2019).

[7] C. E. Dreyer, A. Alkauskas, J. L. Lyons, A. Janotti, and C. G. Van de Walle, "First-principles calculations of point defects for quantum technologies," Annu. Rev. Mater. Res. **48**, 1–26 (2018).

[8] R. Adler, C.-J. Kang, C.-H. Yee, and G. Kotliar, "Correlated materials design: Prospects and challenges," Rep. Prog. Phys. **82**, 012504 (2018).

[9] W. M. C. Foulkes, L. Mitas, R. J. Needs, and G. Rajagopal, "Quantum Monte Carlo simulations of solids," Rev. Mod. Phys. **73**, 33–83 (2001).

[10] R. M. Martin, L. Reining, and D. M. Ceperley, *Interacting Electrons* (Cambridge University Press, 2016).

[11] L. K. Wagner and D. M. Ceperley, "Discovering correlated fermions using quantum Monte Carlo," Rep. Prog. Phys. **79**, 094501 (2016).

[12] R. J. Needs, M. D. Towler, N. D. Drummond, P. López Ríos, and J. R. Trail, "Variational and diffusion quantum Monte Carlo calculations with the CASINO code," J. Chem. Phys. **152**, 154106 (2020).

[13] S. Pilati, E. M. Inack, and P. Pieri, "Self-learning projective quantum Monte Carlo simulations guided by restricted Boltzmann machines," Phys. Rev. E **100**, 043301 (2019).

[14] D. Pfau, J. S. Spencer, A. G. D. G. Matthews, and W. M. C. Foulkes, "*Ab initio* solution of the many-electron Schrödinger equation with deep neural networks," Phys. Rev. Res. **2**, 033429 (2020).

[15] A. Acevedo, M. Curry, S. H. Joshi, B. Leroux, and N. Malaya, "Vandermonde wave function ansatz for improved variational Monte Carlo," in *2020 IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers (DLS)* (IEEE, 2020), pp. 40–47.

[16] J. Hermann, Z. Schätzle, and F. Noé, "Deep-neural-network solution of the electronic Schrödinger equation," Nat. Chem. **12**, 891–897 (2020).

[17] X. Li, Z. Li, and J. Chen, "Ab initio calculation of real solids via neural network ansatz," Nat. Commun. **13**, 7895 (2022).

[18] M. Wilson, N. Gao, F. Wudarski, E. Rieffel, and N. M. Tubman, "Simulations of state-of-the-art fermionic neural network wave functions with diffusion Monte Carlo," arXiv:2103.12570 [physics, physics:quant-ph] (2021).

[19] J. A. R. Shea and E. Neuscamman, "Size consistent excited states via algorithmic transformations between variational principles," J. Chem. Theory Comput. **13**, 6078–6088 (2017).

[20] M. Dash, J. Feldt, S. Moroni, A. Scemama, and C. Filippi, "Excited states with selected configuration interaction-quantum Monte Carlo: Chemically accurate excitation energies and geometries," J. Chem. Theory Comput. **15**, 4896–4906 (2019).

[21] L. Otis, I. M. Craig, and E. Neuscamman, "A hybrid approach to excited-state-specific variational Monte Carlo and doubly excited states," J. Chem. Phys. **153**, 234105 (2020).

[22] L. N. Tran and E. Neuscamman, "Improving excited-state potential energy surfaces via optimal orbital shapes," J. Phys. Chem. A **124**, 8273–8279 (2020).

[23] J. Feldt and C. Filippi, "Excited-state calculations with quantum Monte Carlo," in *Quantum Chemistry and Dynamics of Excited States* edited by L. González and R. Lindh (Wiley, 2020) pp. 247–276.

[24] M. Dash, S. Moroni, C. Filippi, and A. Scemama, "Tailoring CIPSI expansions for QMC calculations of electronic excitations: The case study of thiophene," J. Chem. Theory Comput. **17**, 3426–3434 (2021).

[25] S. Pathak, B. Busemeyer, J. N. B. Rodrigues, and L. K. Wagner, "Excited states in variational Monte Carlo using a penalty method," J. Chem. Phys. **154**, 034101 (2021).

[26] J. T. Krogel, M. Yu, J. Kim, and D. M. Ceperley, "Quantum energy density: Improved efficiency for quantum Monte Carlo calculations," Phys. Rev. B **88**, 035137 (2013).

[27] K. Ryczko, J. T. Krogel, and I. Tamblyn, "Machine learning diffusion Monte Carlo energy densities," arXiv:2205.04547 [cond-mat] (2022).

[28] L. K. Wagner, "Types of single particle symmetry breaking in transition metal oxides due to electron correlation," J. Chem. Phys. **138**, 094106 (2013).

[29] H. J. Changlani, H. Zheng, and L. K. Wagner, "Density-matrix based determination of low-energy model Hamiltonians from *ab initio* wavefunctions," J. Chem. Phys. **143**, 102814 (2015).

[30] H. Zheng, H. J. Changlani, K. T. Williams, B. Busemeyer, and L. K. Wagner, "From real materials to model Hamiltonians with density matrix downfolding," Front. Phys. **6**, 43 (2018).

[31] Y. Chang and L. K. Wagner, "Effective spin-orbit models using correlated first-principles wave functions," Phys. Rev. Res. **2**, 013195 (2020).

[32] A. Zen, S. Sorella, M. J. Gillan, A. Michaelides, and D. Alfè, "Boosting the accuracy and speed of quantum Monte Carlo: Size consistency and time step," Phys. Rev. B **93**, 241118 (2016).

[33] T. A. Anderson and C. J. Umrigar, "Nonlocal pseudopotentials and time-step errors in diffusion Monte Carlo," J. Chem. Phys. **154**, 214110 (2021).

[34] J. Kim, A. D. Baczewski, T. D. Beaudet, A. Benali, M. C. Bennett, M. A. Berrill, N. S. Blunt, E. J. L. Borda, M. Casula, D. M. Ceperley, S. Chiesa, B. K. Clark, R. C. Clay, K. T. Delaney, M. Dewing, K. P. Esler, H. Hao, O. Heinonen, P. R. C. Kent, J. T. Krogel, I. Kylänpää, Y. W. Li, M. G. Lopez, Y. Luo, F. D. Malone, R. M. Martin, A. Mathuriya, J. McMinis, C. A. Melton, L. Mitas, M. A. Morales, E. Neuscamman, W. D. Parker, S. D. Pineda Flores, N. A. Romero, B. M. Rubenstein, J. A. R. Shea, H. Shin, L. Shulenburger, A. F. Tillack, J. P. Townsend, N. M. Tubman, B. Van Der Goetz, J. E. Vincent, D. C. Yang, Y. Yang, S. Zhang, and L. Zhao, "QMCPACK: An open source *ab initio* quantum Monte Carlo package for the electronic structure of atoms, molecules and solids," J. Phys.: Condens. Matter **30**, 195901 (2018).

[35] K. Nakano, C. Attaccalite, M. Barborini, L. Capriotti, M. Casula, E. Coccia, M. Dagrada, C. Genovese, Y. Luo, G. Mazzola, A. Zen, and S. Sorella, "TurboRVB: A many-body toolkit for *ab initio* electronic simulations by quantum Monte Carlo," J. Chem. Phys. **152**, 204121 (2020).

[36] C. Umrigar, Cornell–Holland *ab initio* materials package–CHAMP, https://cyrus.lassp.cornell.edu/champ, 2016.

[37] B. L. Hammond, W. A. Lester, and P. J. Reynolds, *Monte Carlo Methods in Ab Initio Quantum Chemistry*, World Scientific Lecture and Course Notes in Chemistry Vol. 1 (World Scientific, 1994).

[38]M. P. Nightingale and C. J. Umrigar, *Quantum Monte Carlo Methods in Physics and Chemistry* (Springer Science & Business Media, 1998).
[39]I. Prigogine and S. A. Rice, *New Methods in Computational Quantum Mechanics* (John Wiley & Sons, 2009), Vol. 93.
[40]J. Kolorenč and L. Mitas, "Applications of quantum Monte Carlo methods in condensed systems," Rep. Prog. Phys. **74**, 026502 (2011).
[41]B. M. Austin, D. Y. Zubarev, and W. A. Lester, "Quantum Monte Carlo and related approaches," Chem. Rev. **112**, 263–288 (2012).
[42]J. Toulouse, R. Assaraf, and C. J. Umrigar, "Chapter fifteen: Introduction to the variational and diffusion Monte Carlo methods," in *Electron Correlation in Molecules–Ab Initio Beyond Gaussian Quantum Chemistry*, Advances in Quantum Chemistry Vol. 73, edited by P. E. Hoggan and T. Ozdogan (Academic Press, 2016), pp. 285–314.
[43]S. Yuan, Y. Chang, and L. K. Wagner, "Quantification of electron correlation for approximate quantum calculations," J. Chem. Phys. **157**, 194101 (2022).
[44]L. K. Wagner and L. Mitas, "Energetics and dipole moment of transition metal monoxides by quantum Monte Carlo," J. Chem. Phys. **126**, 034105 (2007).
[45]C. J. Umrigar, K. G. Wilson, and J. W. Wilkins, "Optimized trial wave functions for quantum Monte Carlo calculations," Phys. Rev. Lett. **60**, 1719–1722 (1988).
[46]S. Sorella, M. Casula, and D. Rocca, "Weak binding between two aromatic rings: Feeling the van der Waals attraction by quantum Monte Carlo methods," J. Chem. Phys. **127**, 014105 (2007).
[47]L. K. Wagner, M. Bajdich, and L. Mitas, "QWalk: A quantum Monte Carlo program for electronic structure," J. Comput. Phys. **228**, 3390–3404 (2009).
[48]L. Mitáš, E. L. Shirley, and D. M. Ceperley, "Nonlocal pseudopotentials and diffusion Monte Carlo," J. Chem. Phys. **95**, 3467–3475 (1991).
[49]W. L. McMillan, "Ground state of liquid He^4," Phys. Rev. **138**, A442–A451 (1965).
[50]C. Lin, F. H. Zong, and D. M. Ceperley, "Twist-averaged boundary conditions in continuum quantum Monte Carlo algorithms," Phys. Rev. E **64**, 016702 (2001).
[51]S. Chiesa, D. M. Ceperley, R. M. Martin, and M. Holzmann, "Finite-size error in many-body simulations with long-range interactions," Phys. Rev. Lett. **97**, 076404 (2006).
[52]S. Pathak and L. K. Wagner, "A light weight regularization for wave function parameter gradients in quantum Monte Carlo," AIP Adv. **10**, 085213 (2020).
[53]S. Sorella, "Green function Monte Carlo with stochastic reconfiguration," Phys. Rev. Lett. **80**, 4558–4561 (1998).
[54]M. Casula and S. Sorella, "Geminal wave functions with Jastrow correlation: A first application to atoms," J. Chem. Phys. **119**, 6500–6511 (2003).
[55]A. J. Williamson, R. Q. Hood, R. J. Needs, and G. Rajagopal, "Diffusion quantum Monte Carlo calculations of the excited states of silicon," Phys. Rev. B **57**, 12140–12144 (1998).
[56]K. K. Docken and J. Hinze, "LiH potential curves and wavefunctions for $X^1\Sigma^+$, $A^1\Sigma^+$, $B^1\Pi$, $^3\Sigma+$, and 3Π," J. Chem. Phys. **57**, 4928–4936 (1972).
[57]H.-J. Werner and P. J. Knowles, "A second order multiconfiguration SCF procedure with optimum convergence," J. Chem. Phys. **82**, 5053–5063 (1985).
[58]F. Schautz and C. Filippi, "Optimized Jastrow–Slater wave functions for ground and excited states: Application to the lowest states of ethene," J. Chem. Phys. **120**, 10931–10941 (2004).
[59]C. Filippi, M. Zaccheddu, and F. Buda, "Absorption spectrum of the green fluorescent protein chromophore: A difficult case for ab initio methods?," J. Chem. Theory Comput. **5**, 2074–2087 (2009).
[60]A. Cuzzocrea, A. Scemama, W. J. Briels, S. Moroni, and C. Filippi, "Variational principles in quantum Monte Carlo: The troubled story of variance minimization," J. Chem. Theory Comput. **16**, 4203–4212 (2020).
[61]I. Tobias, R. J. Fallon, and J. T. Vanderslice, "Potential energy curves for CO," J. Chem. Phys. **33**, 1638–1640 (1960).
[62]G. Ortiz, D. M. Ceperley, and R. M. Martin, "New stochastic method for systems with broken time-reversal symmetry: 2D fermions in a magnetic field," Phys. Rev. Lett. **71**, 2777–2780 (1993).
[63]R. Assaraf, M. Caffarel, and A. Khelif, "Diffusion Monte Carlo methods with a fixed number of walkers," Phys. Rev. E **61**, 4566–4575 (2000).
[64]M. Calandra Buonaura and S. Sorella, "Numerical study of the two-dimensional Heisenberg model using a Green function Monte Carlo technique with a fixed number of walkers," Phys. Rev. B **57**, 11446–11456 (1998).
[65]D. H. Davis, "Critical-size calculations for neutron systems by the Monte Carlo method," Report No. UCRL-6707, Lawrence Radiation Laboratory, 1961.
[66]M. Casula, "Beyond the locality approximation in the standard diffusion Monte Carlo method," Phys. Rev. B **74**, 161102 (2006).
[67]L. Dalcín, R. Paz, and M. Storti, "MPI for Python," J. Parallel Distrib. Comput. **65**, 1108–1115 (2005).
[68]L. Dalcín, R. Paz, M. Storti, and J. D'Elía, "MPI for Python: Performance improvements and MPI-2 extensions," J. Parallel Distrib. Comput. **68**, 655–662 (2008).
[69]L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, "Parallel distributed computing using Python," Adv. Water Resour. **34**, 1124–1139 (2011).
[70]L. Dalcin and Y.-L. L. Fang, "mpi4py: Status update after 12 years of development," Comput. Sci. Eng. **23**, 47–54 (2021).
[71]Message passing interface forum, MPI: A message-passing interface standard version 4.0, 2021.
[72]Dask development team, Dask: Library for dynamic task scheduling, 2016.
[73]R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, "CuPy: A NumPy-compatible library for NVIDIA GPU calculations," in *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, 4-9 December 2017, (Curran Associates, 2017), p. 7.
[74]M. C. Bennett, C. A. Melton, A. Annaberdiyev, G. Wang, L. Shulenburger, and L. Mitas, "A new generation of effective core potentials for correlated calculations," J. Chem. Phys. **147**, 224106 (2017).
[75]A. Annaberdiyev, G. Wang, C. A. Melton, M. Chandler Bennett, L. Shulenburger, and L. Mitas, "A new generation of effective core potentials from correlated calculations: 3d transition metal series," J. Chem. Phys. **149**, 134108 (2018).
[76]Y. Chang and L. K. Wagner, "Learning emergent models from *ab initio* many-body calculations," arXiv:2302.02899 [cond-mat] (2023).
[77]Opensource.org, https://opensource.org/licenses/MIT; accessed on November 4, 2022.
[78]SPDX workgroup a Linux foundation project, https://spdx.org/licenses/MIT.html (2018); accessed on November 4, 2022.
[79]J. H. Saltzer, "The origin of the 'MIT license,'" IEEE Ann. Hist. Comput. **42**, 94–98 (2020).
[80]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).
[81]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, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," Nat. Methods **17**, 261–272 (2020).
[82]A. Collette, "HDF5 for Python," http://h5py.alfven.org (2008).