JCTC Journal of Chemical Theory and Computation

Article

Cite This: J. Chem. Theory Comput. 2018, 14, 3504−3511

pubs.acs.org/JCTC

Psi4NumPy: An Interactive Quantum Chemistry Programming Environment for Reference Implementations and Rapid Development

Daniel G. A. Smith,*[,†] Lori A. Burns,[†] Dominic A. Sirianni,[†] Daniel R. Nascimento,[‡] Ashutosh Kumar,[¶] Andrew M. James,[¶] Jeffrey B. Schriber,[§] Tianyuan Zhang,[§] Boyi Zhang,[∥] Adam S. Abbott,[∥] Eric J. Berquist,[⊥] Marvin H. Lechner,[#] Leonardo A. Cunha,[□] Alexander G. Heide,[△] Jonathan M. Waldrop,[▽] Tyler Y. Takeshita,[○] Asem Alenaizan,[†] Daniel Neuhauser,[◇] Rollin A. King,[△] Andrew C. Simmonett,[●] Justin M. Turney,[∥] Henry F. Schaefer,[∥] Francesco A. Evangelista,[§] A. Eugene DePrince III,[‡] T. Daniel Crawford,[¶] Konrad Patkowski,[▽] and C. David Sherrill[†]

[†]Center for Computational Molecular Science and Technology, School of Chemistry and Biochemistry, School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0400, United States

[‡]Department of Chemistry and Biochemistry, Florida State University, Tallahassee, Florida 32306-4390, United States

[¶]Department of Chemistry, Virginia Tech, Blacksburg, Virginia 24061, United States

[§]Department of Chemistry, Emory University, Atlanta, Georgia 30322, United States

[∥]Center for Computational Quantum Chemistry, University of Georgia, Athens, Georgia 30602, United States

[⊥]University of Pittsburgh, Pittsburgh, Pennsylvania 15260, United States

[#]Department of Chemistry, Technical University of Munich, 80333 Munich, Germany

[□]The Technical Institute of Aeronautics, São José dos Campos, 12228-900, Brazil

[△]Department of Chemistry, Bethel University, St. Paul, Minnesota 55112, United States

[▽]Department of Chemistry and Biochemistry, Auburn University, Auburn, Alabama 36849, United States

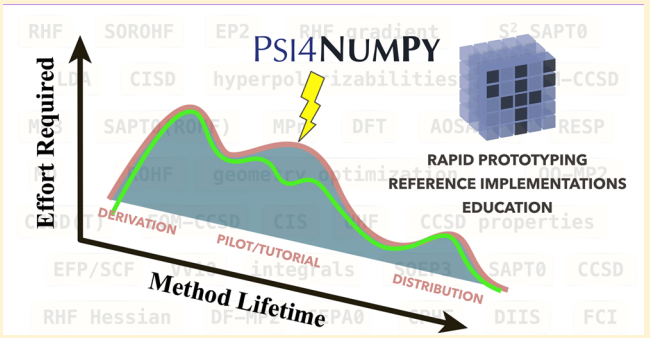
[●]National Institutes of Health - National Heart, Lung and Blood Institute, Laboratory of Computational Biology, 5635 Fishers Lane, T-900 Suite, Rockville, Maryland 20852, United States

[○]Department of Chemistry, University of California Berkeley, Berkeley, California 94720, United States

[◇]Department of Chemistry and Biochemistry, University of California, Los Angeles, California 90095, United States

Supporting Information

ABSTRACT: Psi4NumPy demonstrates the use of efficient computational kernels from the open-source Psi4 program through the popular NumPy library for linear algebra in Python to facilitate the rapid development of clear, understandable Python computer code for new quantum chemical methods, while maintaining a relatively low execution time. Using these tools, reference implementations have been created for a number of methods, including self-consistent field (SCF), SCF response, many-body perturbation theory, coupled-cluster theory, configuration interaction, and symmetry-adapted perturbation theory. Furthermore, several reference codes have been integrated into Jupyter notebooks, allowing background, underlying theory, and formula information to be associated with the implementation. Psi4NumPy tools and associated reference implementations can lower the barrier for future development of quantum chemistry methods. These implementations also demonstrate the power of the hybrid C++/Python programming approach employed by the Psi4 program.



1. INTRODUCTION

The inherent computational expense of most quantum chemical (QC) methods creates substantial pressure for highly optimized implementations. This is a challenge for ongoing research in quantum chemistry as new theoretical methods are typically complex and nontrivial to implement correctly. Fundamentally, computationally efficient codes require a low-

Received: March 21, 2018
Published: May 17, 2018

DOI: 10.1021/acs.jctc.8b00286
J. Chem. Theory Comput. 2018, 14, 3504−3511

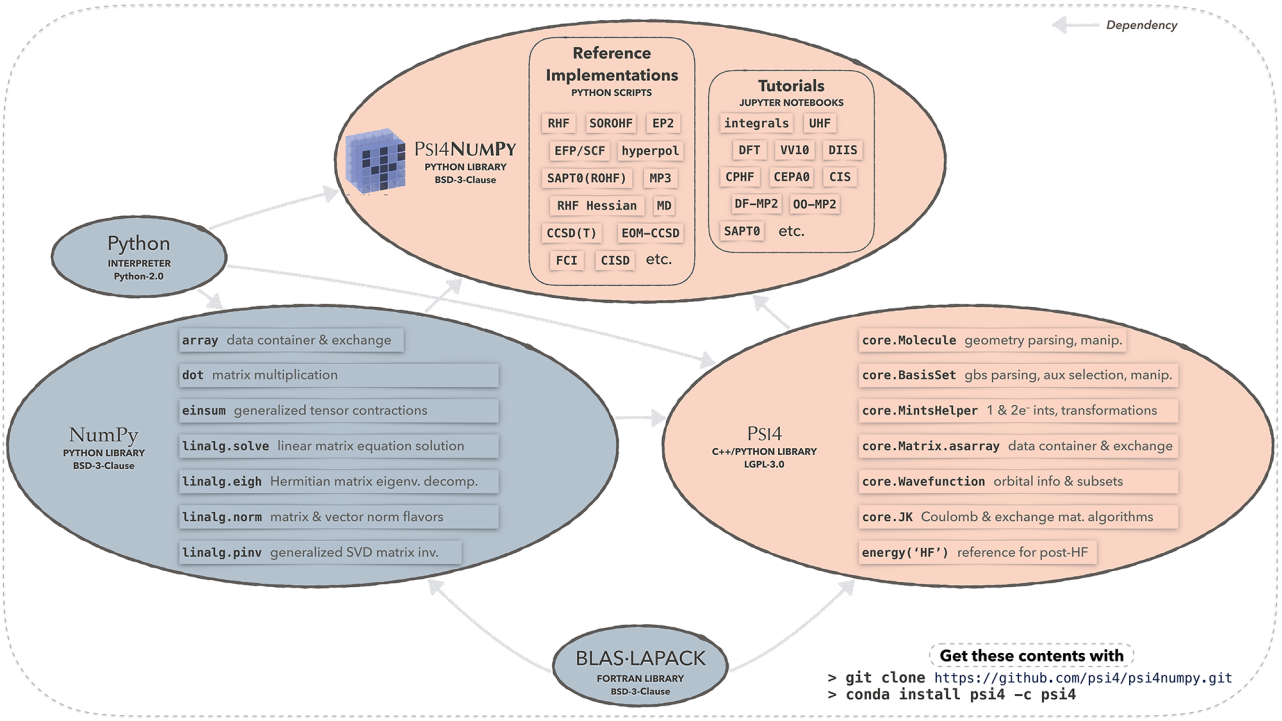


Figure 1. Psi4NumPy draws linear algebra tools from NumPy and fundamental quantum chemistry structures from Psi4 to bring together a practical and convenient environment for code development, verification, and exploration. The most important data structures and functions are shown for NumPy and Psi4 as well as representative tutorial and reference implementations presently in Psi4NumPy.

level programming language like C, C++, or Fortran and several stages of code profiling, testing, and optimization to reach production quality. Therefore, a method's first implementation is typically a messy computer program that is further convoluted over the years by the demands of novel architecture and expansion of features. Additionally, development is often carried out by graduate students not yet proficient in programming, resulting in unconventional coding styles. Future researchers seeking to extend or enhance a method previously developed in-house are often faced with the daunting prospect of deciphering a quite complex existing code.

Still more challenging is implementing or extending an existing method sourced solely from the literature. Often, a paper describing a new quantum chemical method that properly focuses on scientific detail falls short on algorithmic or numerical detail sufficient for independent reimplementation. Indeed, methods are so complex that the original equations frequently include typos, which are generally tracked through institutional lore rather than published errata. Additionally, modern approaches often employ combinations of approximations with multiple numerical cutoffs, exacerbating the reproducibility problem. This paradigm is illustrated within a recent comment,[1] whereby several corrections to equations originally published in 2011 for a two-level semiempirical method[2] were proposed after being re-engineered to reproduce values computed using a binary program distributed with the original publication. Even facilitated through private communication with the method's author, this cycle of rediscovery and reimplementation is both highly nontrivial and unsustainable. In the specific case of ref 2, fortunately, an open-source program[3] has been made available by the commenting author that implements the method and proposed changes, so that further extensions of the method can proceed with this program as a reference.

Such reference implementations (easy-to-read, unoptimized computer programs solely targeting the correct result) can be a helpful initial step toward developing or understanding a complex method, yet they are not widely available in quantum chemistry. To our knowledge, reference implementations and benchmarking have only been performed in a large-scale way for density functional theory (DFT) exchange-correlation kernels[4] and periodic boundary condition DFT with pseudopotentials.[5] One factor limiting more widespread use of reference implementations for quantum chemistry is that methods are often so computationally demanding that a basic, unoptimized implementation is too slow for computations on even the smallest molecules. What is needed is an alliance of a QC code that is easy to peruse and manipulate with underlying non-QC routines that are fast enough for testing on nontrivial molecules.

Here we present Psi4NumPy, a framework for the creation of clear, readable reference implementations of quantum chemical methods and for the rapid development of new methods. Psi4NumPy takes advantage of Psi4's[6] application programming interface (API) that makes efficient computational kernels written in C++ available from Python, a language that is easy to learn and has become very popular in scientific computing. As a high-level language, Python allows complex tasks to be specified with relatively few lines of code. Psi4NumPy capitalizes on the straightforward conversion of Psi4 tensors to NumPy,[7] a numerical linear algebra package and array manipulation tool. NumPy's own low-level back-end is written in C to ensure that all data arrays can use the optimized Basic Linear Algebra Subprograms (BLAS) library[8] for common linear algebra operations. For working with arrays in Python, NumPy provides greater efficiency over native list and array representations. In addition, NumPy's tensor syntax allows many operations to be completed without writing "for" loops, leading to more concise

DOI: 10.1021/acs.jctc.8b00286
J. Chem. Theory Comput. 2018, 14, 3504−3511

code. Psi4NumPy has been packaged for minimal setup, requiring only several minutes, with no preinstalled compilers necessary on 64-bit Linux, Mac, and Windows. Here we introduce the main elements of the Psi4NumPy framework and illustrate them with a substantial collection of reference implementations for standard quantum chemical methods and numerical techniques. Psi4NumPy is built entirely on Free and Open Source Software (FOSS)[9] as shown in Figure 1 to ensure a barrierless entry to quantum chemistry programming.

There have been a number of tensor libraries for quantum chemistry that also provide code for tensor contractions that are easy to read and that follow the structure of the original equations, including the Tensor Contraction Engine,[10] the Cyclops Tensor Framework,[11] and LibTensor.[12] These libraries may be more optimal than NumPy for large-scale tensor operations, and, indeed, many of them also work for distributed-parallel computing. However, we believe NumPy is a better choice for reference implementations and rapid prototyping because it is broadly familiar in scientific computing, can be installed using canonical Python package managers already available on user systems, and does not require compilation so that users can make changes and evaluate the results in a matter of seconds. Along similar lines to Psi4NumPy, the PySCF[13] package also employs NumPy and interfaces to C and C++ packages. However, PySCF is a quantum chemistry package, rather than independent reference implementations, and leans toward performance over readability. The Psi4NumPy project explicitly makes the choice to prioritize readable and clear code with the understanding that the code will be nonoptimal for any operations beyond those that do make use of Psi4's internal routines for computationally intensive operations.

Several of the reference implementations have been augmented by tutorial-style introductions to the relevant theory. The Psi4NumPy tutorial collection includes self-consistent field (SCF), DFT,[14] many-body perturbation theory (MBPT),[15] symmetry-adapted perturbation theory (SAPT),[16,17] coupled-cluster (CC),[18] and configuration interaction (CI)[19,20] theories, with additional sections detailing the theory and implementation of linear response, geometry optimizations, and Verlet integrators. It is our hope that Psi4NumPy and the accompanying reference code will lower the barrier to implementing and understanding quantum chemical methods.

Shortly before submission, the authors chanced upon the Quantum Chemistry Program Exchange (QCPE),[21] whose goals of software (particularly self-contained software) accessibility, algorithm explication, and free software "publishing" Psi4NumPy shares. The general tools embraced by Psi4NumPy (GitHub for communication, NumPy for linear algebra, Python for interfacing, and Jupyter for illumination) further allow rapid prototyping and a gentle learning curve. As part of Psi4NumPy's FOSS philosophy, we actively encourage the community to submit new theories to the open GitHub repository through canonical GitHub pull-request processes. Contributions must minimally provide some metadata, some checks to ensure the submitted code is correct, and be buildable and testable through a continuous integration service. Otherwise, contributions can be as independent of or reliant upon Psi4 and NumPy as authors need to illustrate their approach. In this manner, Psi4NumPy can be thought of as a modern successor to QCPE built to serve the flexible needs of the community.

2. BASIC TOOLS

The basic premise of Psi4NumPy is to leverage Psi4 to generate quantum chemistry-specific quantities and the NumPy library[7] for all other tensor manipulations. The latest version of Psi4 (version 1.1; May 2017) has added the option to import Psi4 as a Python module. In this way, both the Psi4 and NumPy libraries can be loaded into a single Python script and used in cooperation.

A key capacity in this enterprise is seamless translation between the NumPy and Psi4 data classes. For example, converting from a NumPy array to a Psi4 matrix and back again can be easily accomplished:

```
import numpy, psi4
np_array = numpy.zeros((5, 5))
psi4_matrix = psi4.core.Matrix.from_array(np_array)
new_np_array = numpy.array(psi4_matrix)
```
(1)

At the core of this procedure is NumPy's `array_interface`[22] protocol, a basic specification for dense matrices primarily consisting of

1. the starting memory location for an in-memory array
2. the overall "shape" of the array [$(n,)$ for a vector, (n, m) for a matrix, etc.]
3. the type of data involved (`double64`, `int32`, etc.)

This specification is compact and widely used by the scientific Python community, including by SciPy[23] for a numerical integration and optimization, Dask[24] for distributed computing, and Tensorflow[25] for GPU tensor operations. Using the `array_interface`, it becomes straightforward to allow NumPy access to Psi4 data classes, enabling both Psi4 and NumPy to access and manipulate the same data. For example, the statement below will overwrite the data of the Psi4 Matrix class in place with a random NumPy array:

```
psi4_matrix.np[:] = numpy.random.rand(5, 5)
```
(2)

In this way, the typical separation between general tensor frameworks and custom quantum chemistry data structures is removed.

A description of the full set of capabilities of the `array_interface` is available in the Psi4 documentation: http://psicode.org/psi4manual/master/numpy.html.

2.1. `Wavefunction` Objects. In Psi4 all built-in methodologies have the option to return a `Wavefunction` object that holds basic information about the previous computation or, in some cases, holds functions for readily computing advanced quantities. Obtaining the `Wavefunction` object in this manner is straightforward:

```
mol = psi4.geometry("""
0
H 1 0.96
H 1 0.96 2 104.5
""")

hf_e, hf_wfn = psi4.energy("HF/cc-pVDZ", molecule=mol,
    return_wfn=True)
```
(3)

Once a `Wavefunction` object is obtained, a variety of attributes can be queried using standard Python syntax:

DOI: 10.1021/acs.jctc.8b00286
J. Chem. Theory Comput. 2018, 14, 3504−3511

```
# Number of doubly occupied orbitals
docc = hf_wfn.ndocc()

# Alpha orbital coefficient matrix
Ca = hf_wfn.Ca()

# Occupied subset of the alpha orbitals
Ca_occ = hf_wfn.Ca_subset("AO", "OCC")
```
(4)

In addition to generating useful information after a computation, a `Wavefunction` object can also be passed as reference state to a further computation. For Psi4NumPy, this means that reference implementations of post-Hartree−Fock methods (MPn, CCSD, etc.) need not (but may) recode their own Hartree−Fock program as all required quantities are available from Psi4 through production-quality routines. This simultaneously reduces code duplication and increases readability, both of which are cornerstones of the Psi4NumPy project.

2.2. Integrals. Psi4 offers a wide selection of efficient C++ tools accessible directly in Python. These tools are largely object-based and capable of storing quantities in memory or on disk. One such object is the `libmints`[6] library, which is currently the primary interface for computing one- and two-electron integrals in Psi4. This library is accessible through the `MintsHelper` class that directs the efficient computation and storage of molecular integrals Python-side:

```
# Create instance of MintsHelper using primary basis set
mints = psi4.core.MintsHelper(primary_basis)

# Compute one-electron AO overlap matrix
S = mints.ao_overlap()

# Compute core Hamiltonian matrix
T = mints.ao_kinetic()
V = mints.ao_potential()
H = T + V

# Compute two-electron integrals in AO basis in memory
I_ao = mints.ao_eri()
```
(5)

Each of the above `MintsHelper` class methods returns a Psi4 matrix which can be converted to a NumPy array using `numpy.asarray(matrix)` or modified in place with the `matrix.np` accessor.

In addition to computing molecular integrals, the `libmints` library also performs optimized electron repulsion integral (ERI) transformations. For example, the $O(N^5)$ transformation of the two-electron integrals between the atomic orbital and molecular orbital basis is given by

$$(ia|jb) = [[C_{\mu i}[C_{\nu a}(\mu\nu|\lambda\sigma)]]C_{\lambda j}]C_{\sigma b} \tag{6}$$

with Greek letters labeling AOs and Latin letters labeling MOs (i, j—occupied, a, b—virtual) and the Einstein summation convention assumed in eq 6 and throughout the text. This transformation can be performed easily with

```
# Occupied and virtual subsets of SCF orbital coefficient matrices
Ca_occ = hf_wfn.Ca_subset("AO", "OCC")
Ca_virt = hf_wfn.Ca_subset("AO", "VIR")

# AO basis to MO basis in-memory ERI transform
I_mo = mints.mo_transform(Ca_occ, Ca_virt, I_ao, Ca_occ, Ca_virt)
```
(7)

In this manner, arbitrary ERI transformations may be performed, allowing both speed and flexibility for constructing reference implementations.

2.3. Coulomb and Exchange (JK) Matrix Objects. A key component in SCF-level theories is the contraction of the 4-index electron repulsion integrals with the 2-index density matrix to form J and K matrices:

$$J_{\lambda\sigma}[D] \equiv (\lambda\sigma|\mu\nu)D_{\mu\nu} \tag{8}$$

$$K_{\lambda\sigma}[D] \equiv (\lambda\mu|\sigma\nu)D_{\mu\nu} \tag{9}$$

Psi4 provides objects for computing Coulomb (J) and Exchange (K) matrices, with specialized algorithms for integral-direct, PK supermatrix,[26] or density fitting (DF) scenarios. For the *DF−JK* object, it is often advantageous to use a factorized form of the density matrix

$$D_{\mu\nu} \equiv \sum_p C_{\mu p}^{\text{left}} C_{\nu p}^{\text{right}} \tag{10}$$

where p is a general MO index. For example, in canonical Restricted Hartree−Fock (RHF), the density matrix takes the form

$$D_{\mu\nu}^{\text{RHF}} = \sum_i C_{\mu i} C_{\nu i} \tag{11}$$

where i runs only over occupied orbitals. The computation of the RHF JK matrices can be translated directly to Python code with the following lines:

```
# Create a JK object in the current primary basis set
jk = psi4.core.JK.build(primary_basis)

# Add the occupied parts of the SCF orbital matrix
jk.add_C_left(C_occupied)
jk.add_C_right(C_occupied)

# Perform the computation and obtain the J and K matrices
jk.compute()
J = jk.J()
K = jk.K()
```
(12)

In this fashion, virtually any SCF-level theory can be formulated at the Psi4NumPy layer by handling only 2-D arrays with NumPy (typically by threaded vendor BLAS) and leaving the 3- and 4-D arrays to Psi4 libraries (using optimized C++ routines). Additional examples that can be written with *JK* matrices are coupled-perturbed SCF, time-dependent SCF, SCF stability analysis, and most terms found in symmetry-adapted perturbation theory besides dispersion-like quantities. Thus, SCF-level theories can be implemented with the same efficiency as their pure C++ counterparts.

To illustrate this point, the Psi4 SCF program is compared against a Psi4NumPy implementation on an Intel i7-5930K processor with the adenine·thymine complex in the aug-cc-pVTZ basis (1127 basis functions) using a *DF−JK* build on six cores. The Psi4 SCF program took 250 s while the Psi4NumPy implementation took 245 s. This should not be surprising as each computation spent 94% of the total wall time computing the J and K quantities (both implementations used 18 SCF iterations), and all other operations of nonnegligible cost use the same BLAS implementations.

3. RAPID DEVELOPMENT

A key objective of the Psi4NumPy framework is to provide an easy-to-use development environment for rapid prototyping. Vital to this goal is NumPy's `einsum` function that performs arbitrary tensor contractions using Einstein summation syntax. The `einsum` syntax first requires a string of the indices of contraction followed by the NumPy arrays involved in the `einsum` expression. For example, the atomic orbital to

DOI: 10.1021/acs.jctc.8b00286
J. Chem. Theory Comput. 2018, 14, 3504−3511

molecular orbital 4-index transformation of eq 6 and code snippet (7) could be accomplished by

```
I_mo = numpy.einsum("pi,qa,pqrs,rj,sb->iajb",
                    Ca_occ, Ca_virt, I_ao,
                    Ca_occ, Ca_virt)
```
(13)

Recently, one of us (D.G.A.S.) modified NumPy's `einsum` function so that it will automatically factorize the incoming tensor expression to reduce the cost of the operation from naive N^8 to the conventional N^5 version. This feature is available in NumPy 1.12 and onward, with additional optimizations and BLAS usage occurring in NumPy 1.14. In addition, a drop-in replacement for the `einsum` function, which makes optimal use of vendor BLAS, can be found through the Optimized Einsum project.[27]

Using the `einsum` function, it is straightforward to transcribe existing equations directly into working code without a compilation stage. While the resulting program is not as efficient for post-SCF level theories as a full implementation in a low-level language, the code is easy to read and modify without the need for compilation, allowing considerable flexibility when prototyping. In addition, the resulting program will provide correct answers for the given expressions, sparing the developer any worry whether low-level code is correct.

As an example of rapid prototyping, we consider an intermediate quantity appearing in the CCSD amplitude residual equations.[28] For virtual indices a, b, c, d and occupied indices i, j, k, eq 8 of ref 28 is written as

$$W_{jaci} = \langle ja||ci\rangle + t_i^d\langle ja||cd\rangle - t_k^a\langle jk||ci\rangle - \left(\frac{1}{2}t_{ik}^{da} + t_i^d t_k^a\right)\langle jk||cd\rangle$$

which can be directly translated into a function:

```
def build_Wjaci(T1, T2, MO):
    Wjaci = MO[o, v, v, o].copy()
    Wjaci += numpy.einsum("id,jacd->jaci", T1, MO[o, v, v, v])
    Wjaci -= numpy.einsum("ka,jkci->jaci", T1, MO[o, o, v, o])

    tmp = 0.5 * T2 + numpy.einsum("id,ka->ikda", T1, T1)

    Wjaci -= numpy.einsum("ikda,jkcd->jaci", tmp, MO[o, o, v, v])
    return Wjaci
```
(14)

Here, `MO` holds the 4-index antisymmetrized integrals, `T1` and `T2` the current amplitudes, and the `o`, `v` quantities are Python-based slices so that `MO[o, v, v, v]` returns the occupied–virtual–virtual–virtual block of the antisymmetrized integrals.

To our knowledge, the first implementations of symmetry-adapted perturbation theory with complete active space SCF references [SAPT(CASSCF)], fourth-order electron propagator theory, and transcorrelated theories have all been achieved using these rapid prototyping techniques.

4. ACCESS AND CONTRIBUTIONS

To ensure ease of community access to the Psi4NumPy project, all software dependencies are made available as binary Conda packages[29] either by us (e.g., Psi4) or by Anaconda or Intel (e.g., NumPy, Matplotlib, Jupyter). Through this route, binary distributions are installable in a single line to all common computing platforms, so users are not required to compile, link against the correct libraries, or debug runtime issues. We hope that the ready accessibility of these tools facilitates their use in methods development and in the creation of additional publicly available reference implementations.

To lower the barrier to contribution, guidance is included in the repository regarding attribution, citations, and testing. Though the authors adhere to Python software development best practices in their other projects, they avoid advanced Python syntax, organization, file linking, or other jargon-ized code in Psi4NumPy in favor of straightforward scripts and Jupyter notebooks for ease of community involvement. Educators are encouraged to base lessons and laboratories upon this work and are also referred to the Psi4Education project.[30]

5. REFERENCE IMPLEMENTATIONS

To illustrate the Psi4NumPy tools and to provide a resource to the quantum chemistry methods development community, we have created a number of reference implementations and made them publicly available on GitHub at https://github.com/psi4/psi4numpy. We intend to add to this collection over time. Given the wide spectrum of quantum chemical methods, we also encourage submissions from other developers.

The Psi4NumPy reference implementations, while not necessarily as efficient as optimized versions in a low-level language, furnish at least the basic requirements for a programmer to reproduce the methodology. These references provide a medium to explain minute details that might not be included in a corresponding paper and to record algorithmic tricks used to improve numerical stability or computational efficiency. In addition, these clear implementations will make explicit any important steps that might not be mentioned in a paper because they are assumed to be background knowledge in a given subfield of quantum chemistry.

Programmers can use these reference implementations to obtain intermediate quantities to validate a new implementation at every step, ensuring accuracy and assisting in the process of debugging a new program. These reference implementations can also be used as starting points for either building upon existing methodologies or exploring new methodologies in combination with the rapid prototyping aspects of this project.

Current reference implementations include

1. Self-Consistent Field
   (a) Restricted simple and DIIS[31]-accelerated Hartree–Fock
   (b) Restricted, Unrestricted, and Restricted Open-Shell Hartree–Fock
   (c) Restricted, Unrestricted, and Restricted Open-Shell Hartree–Fock time-independent orbital Hessians
   (d) Restricted time-dependent Hartree–Fock and coupled-perturbed Hartree–Fock for dipole hyperpolarizabilities and polarizabilities

DOI: 10.1021/acs.jctc.8b00286
J. Chem. Theory Comput. 2018, 14, 3504–3511

The Fock matrix, **F**, has elements $F_{\mu\nu}$ given (in the atomic orbital basis) as

$$F_{\mu\nu} = H_{\mu\nu} + 2(\mu\nu|\lambda\sigma)D_{\lambda\sigma} - (\mu\lambda|\nu\sigma)D_{\lambda\sigma},$$

where $D_{\lambda\sigma}$ is an element of the one-particle density matrix **D**, constructed from the orbital coefficient matrix **C**:

$$D_{\lambda\sigma} = C_{\sigma i}C_{\lambda i}$$

Using the above equations, the Fock matrix can be constructed as:

```
In [1]: D = numpy.einsum('pi,qi->pq', Cocc, Cocc)

        J = numpy.einsum('pqrs,rs->pq', I, D)
        K = numpy.einsum('prqs,rs->pq', I, D)

        F = H + 2 * J - K
```

Figure 2. Extract from a Jupyter notebook demonstrating the construction of a SCF Fock matrix where I is the 4-index electron repulsion integral array and Cocc is the occupied orbital matrix.

   (e) Restricted Hartree−Fock nuclear gradients and Hessians
2. Many-Body Perturbation Theory
   (a) Canonical and density-fitted MP2
   (b) Spin-integrated and spin−orbital MP3
   (c) Arbitrary-order MP
   (d) Stochastic-orbital RI-MP2[32]
3. Coupled-Cluster
   (a) Simple and DIIS-accelerated CCSD
   (b) CCSD(T)
   (c) CCSD linear response (dipole polarizabilities, optical rotation)
   (d) Time-dependent equation-of-motion CCSD
4. Configuration Interaction
   (a) Excited-state CIS
   (b) Canonical and Davidson−Liu CISD
   (c) Full configuration interaction
5. Symmetry-Adapted Perturbation Theory
   (a) Restricted and Restricted Open-Shell SAPT0
   (b) Atomic orbital implementation of SAPT0
   (c) SAPT0 without the single exchange approximation
6. Electron Propagator Theory
   (a) Spin-integrated and spin−orbital EP2
   (b) Spin−orbital EP3
7. Restrained Electrostatic Potential (RESP) Charge Fitting

5.1. Jupyter Notebook Integration. As a service to the community, some of the reference implementations have been augmented by additional, tutorial-style background information on various subfields of quantum chemistry. We found it convenient to add this additional information using the Jupyter notebook web application,[33] a popular integrated development environment (IDE) for interactive computing in several programming languages that is starting to be adopted by chemists.[34] This IDE allows code to be separated into blocks that can be recomputed dynamically so that users can work on each fundamental part of a new code or tutorial at a time without needing to recompute all quantities before that point. An example part of the restricted Hartree−Fock notebook can be found in Figure 2.

These documents may be unique within quantum chemistry in that they focus not only on theoretical considerations but also on the details of a method's implementation, such as *why* certain programming choices were made. For example, the comparison between a general matrix inversion and solving a set of linear equations demonstrates instability issues that often plague the former technique. Such illustrations should make the Jupyter implementations useful both to new users in quantum chemistry and to experienced users interested in exploring new subfields.

Current tutorial-style Jupyter reference implementations include the following:

1. Introductions to the Psi4NumPy methodology
2. Introduction to Hartree−Fock, DIIS, and density fitting
3. Density Functional Theory: grids, LDA kernels, VV10 dispersion, and asymptotic corrections
4. Møller−Plesset Perturbation Theory: canonical and density-fitted reference implementations of MP2
5. Molecular Properties: Integrals, CPHF, CIS
6. Symmetry-Adapted Perturbation Theory: Canonical and atomic orbital SAPT0 algorithms
7. Orbital-Optimized Methods: OMP2
8. Coupled-Cluster Approximations: CEPA0, CCD
9. Geometry Optimization Techniques: Internal Coordinates, Hessian guesses, and advanced Newton−Raphson methods

Molecular-dynamics tutorials include the following:

1. Periodic Lennard-Jones simulation with Verlet integrators
2. Periodic Ewald electrostatic summation

6. CONCLUSIONS

We believe that the benefits of the Psi4NumPy framework to the computational chemistry community are threefold. Beginning researchers can use the Psi4NumPy reference implementations for *education*. Reference implementations convey not just the underlying mathematical formulas of a given theory but also how to implement these formulas in a manner that avoids common pitfalls such as ill-conditioned numerical equations. Psi4NumPy is likely the most interactive educational resource available in this field: thanks to the Jupyter Notebook format, the learners can explore the implementation step by step and easily try out various modifications and additional approximations.

More advanced researchers who need to reimplement and/or modify a given computational chemistry approach can use the Psi4NumPy reference implementations for *validation*, taking advantage of the code that, thanks to the extensive use of the NumPy `einsum` functionality, provides a nearly one-to-one correspondence between the terms in a formula and the lines of Python code. As a result, it is trivial to switch off, for debugging purposes, any subset of terms as well as generate an arbitrary

DOI: 10.1021/acs.jctc.8b00286
J. Chem. Theory Comput. 2018, 14, 3504−3511

intermediate without even recompiling any code. This feature should be contrasted with the situation when one tries to validate their code against a C++/Fortran implementation from an established electronic-structure package. Once the relevant fragment of code that does the actual computation is found (which is not always trivial), various terms are typically combined in nontrivial ways to improve computational performance. As a result, getting out a specific intermediate for checking the implementation in progress often requires substantive changes to the reference code, not to mention its recompilation. In addition, we include the programmed formulas together with their implementation in the Jupyter Notebooks to alleviate difficulties associated with incompatible notation or even errors in the originally published expressions.

Finally, for researchers who want to develop new functionality, Psi4NumPy is a highly valuable platform for *initial implementation* that is efficient enough for meaningful testing, quick to generate, easy to debug, and has limited opportunities for programming errors. All underlying quantum-chemistry building blocks such as integrals, orbitals, density matrices, and CI vectors are efficiently computed by Psi4 and readily imported in the NumPy format. In particular, a Psi4NumPy implementation of any one-electron theory such as HF or DFT is already close to optimal as the most expensive operations are all written in terms of generalized Coulomb and exchange matrices which are supplied by Psi4. Some of us, together with collaborators, have already taken advantage of the Psi4NumPy capabilities to rapidly generate pilot implementations of brand new electronic-structure approaches.

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge on the ACS Publications website at DOI: 10.1021/acs.jctc.8b00286.

Python reference implementations and tutorials associated with Psi4NumPy 1.0 (ZIP)

■ AUTHOR INFORMATION

Corresponding Author

*E-mail: dgasmith@vt.edu.

ORCID

Daniel G. A. Smith: 0000-0001-8626-0900
Lori A. Burns: 0000-0003-2852-5864
Dominic A. Sirianni: 0000-0002-6464-0213
Daniel R. Nascimento: 0000-0002-2126-8378
Eric J. Berquist: 0000-0001-8186-9522
Tyler Y. Takeshita: 0000-0003-0067-2846
Asem Alenaizan: 0000-0002-0871-664X
Rollin A. King: 0000-0002-1173-4187
Andrew C. Simmonett: 0000-0002-5921-9272
Justin M. Turney: 0000-0003-3659-0711
Henry F. Schaefer: 0000-0003-0252-2083
Francesco A. Evangelista: 0000-0002-7917-6652
A. Eugene DePrince III: 0000-0003-1061-2521
T. Daniel Crawford: 0000-0002-7961-7016
Konrad Patkowski: 0000-0002-4468-207X
C. David Sherrill: 0000-0002-5570-7666

Notes

The authors declare no competing financial interest.

Documents reproducing all currently available reference implementations and interactive tutorials are available free of charge via the Internet at https://zenodo.org/record/1248189. For all future materials, please see https://github.com/psi4/psi4numpy.

■ ACKNOWLEDGMENTS

This work was supported in part by the U.S. National Science Foundation through grants ACI-1449723 and CHE-1566192 to C.D.S; CHE-1661604 to H.F.S; CHE-1554354 to A.E.D.; and CAREER award CHE-1351978 to K.P. B.Z. and T.Y.T.'s contributions to this work were also supported by a Software Fellowship from the Molecular Sciences Software Institute, which is funded by the U.S. National Science Foundation (ACI-1547580). M.H.L. acknowledges financial support by the Studienstiftung des Deutschen Volkes. A.A. was supported jointly by the National Science Foundation and the NASA Astrobiology Program, under the NSF Center for Chemical Evolution, CHE-1504217. F.A.E acknowledges support by the U.S. Department of Energy under Award No. DE-SC0016004 and by a Research Fellowship of the Alfred P. Sloan Foundation. The development of the stochastic orbital techniques was supported in part by the National Science Foundation, grants CHE-1465064 and DMR-1611382.

■ REFERENCES

(1) Briling, K. R. Comment on "A new parametrizable model of molecular electronic structure" [J. Chem. Phys. 135, 134120 (2011)]. *J. Chem. Phys.* **2017**, *147*, 157101.

(2) Laikov, D. N. A new parametrizable model of molecular electronic structure. *J. Chem. Phys.* **2011**, *135*, 134120.

(3) Source code accompanying the comment [K. R. Briling, J. Chem. Phys. 147, 157101(2017)]. https://github.com/briling/qm (accessed September 20th, 2017).

(4) Density Functional Repository; Quantum Chemistry Group, CCLRC Daresbury Laboratory, Daresbury, Cheshire, WA4 4AD United Kingdom. http://www.cse.scitech.ac.uk/ccg/dft/ (accessed September 11, 2017).

(5) Lejaeghere, K.; Bihlmayer, G.; Björkman, T.; Blaha, P.; Blügel, S.; Blum, V.; Caliste, D.; Castelli, I. E.; Clark, S. J.; Dal Corso, A.; de Gironcoli, S.; Deutsch, T.; Dewhurst, J. K.; Di Marco, I.; Draxl, C.; Dułak, M.; Eriksson, O.; Flores-Livas, J. A.; Garrity, K. F.; Genovese, L.; Giannozzi, P.; Giantomassi, M.; Goedecker, S.; Gonze, X.; GrÅnäs, O.; Gross, E. K. U.; Gulans, A.; Gygi, F.; Hamann, D. R.; Hasnip, P. J.; Holzwarth, N. A. W.; Iuşan, D.; Jochym, D. B.; Jollet, F.; Jones, D.; Kresse, G.; Koepernik, K.; Küçükbenli, E.; Kvashnin, Y. O.; Locht, I. L. M.; Lubeck, S.; Marsman, M.; Marzari, N.; Nitzsche, U.; Nordström, L.; Ozaki, T.; Paulatto, L.; Pickard, C. J.; Poelmans, W.; Probert, M. I. J.; Refson, K.; Richter, M.; Rignanese, G.-M.; Saha, S.; Scheffler, M.; Schlipf, M.; Schwarz, K.; Sharma, S.; Tavazza, F.; Thunström, P.; Tkatchenko, A.; Torrent, M.; Vanderbilt, D.; van Setten, M. J.; Van Speybroeck, V.; Wills, J. M.; Yates, J. R.; Zhang, G.-X.; Cottenier, S. Reproducibility in density functional theory calculations of solids. *Science* **2016**, *351*, aad3000.

(6) Parrish, R. M.; Burns, L. A.; Smith, D. G. A.; Simmonett, A. C.; DePrince, A. E.; Hohenstein, E. G.; Bozkaya, U.; Sokolov, A. Y.; Di Remigio, R.; Richard, R. M.; Gonthier, J. F.; James, A. M.; McAlexander, H. R.; Kumar, A.; Saitow, M.; Wang, X.; Pritchard, B. P.; Verma, P.; Schaefer, H. F.; Patkowski, K.; King, R. A.; Valeev, E. F.; Evangelista, F. A.; Turney, J. M.; Crawford, T. D.; Sherrill, C. D. Psi4 1.1: An Open-Source Electronic Structure Program Emphasizing Automation, Advanced Libraries, and Interoperability. *J. Chem. Theory Comput.* **2017**, *13*, 3185−3197.

(7) van der Walt, S.; Colbert, S. C.; Varoquaux, G. The NumPy Array: A Structure for Efficient Numerical Computation. *Comput. Sci. Eng.* **2011**, *13*, 22−30.

DOI: 10.1021/acs.jctc.8b00286
J. Chem. Theory Comput. 2018, 14, 3504−3511

(8) Blackford, L. S.; Demmel, J.; Dongarra, J.; Duff, I.; Hammarling, S.; Henry, G.; Heroux, M.; Kaufman, L.; Lumsdaine, A.; Petitet, A.; Pozo, R.; Remington, K.; Whaley, R. C. An Updated Set of Basic Linear Algebra Subprograms (BLAS). *ACM Trans. Math. Soft.* **2002**, *28*, 135−151.
(9) Open Source Initiative. https://opensource.org/osd (accessed November 28th, 2017).
(10) Baumgartner, G.; Auer, A.; Bernholdt, D. E.; Bibireata, A.; Choppella, V.; Cociorva, D.; Gao, X.; Harrison, R. J.; Hirata, S.; Krishnamoorthy, S.; Krishnan, S.; Lam, C.-C.; Lu, Q.; Nooijen, M.; Pitzer, R. M.; Ramanujam, J.; Sadayappan, P.; Sibiryakov, A. Synthesis of High-Performance Parallel Programs for a Class of ab Initio Quantum Chemistry Models. Proc. *Proc. IEEE* **2005**, *93*, 276−292.
(11) Solomonik, E.; Matthews, D.; Hammond, J. R.; Stanton, J. F.; Demmel, J. A massively parallel tensor contraction framework for coupled-cluster computations. *J. Parallel Distrib. Comput.* **2014**, *74*, 3176−3190.
(12) Manzer, S.; Epifanovsky, E.; Krylov, A. I.; Head-Gordon, M. A General Sparse Tensor Framework for Electronic Structure Theory. *J. Chem. Theory Comput.* **2017**, *13*, 1108−1116.
(13) Sun, Q.; Berkelbach, T. C.; Blunt, N. S.; Booth, G. H.; Guo, S.; Li, Z.; Liu, J.; McClain, J. D.; Sayfutyarova, E. R.; Sharma, S.; Wouters, S.; Chan, G. K. PySCF: the Python-based simulations of chemistry framework. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2018**, *8*, e1340.
(14) Parr, R. G.; Yang, W. *Density-Functional Theory of Atoms and Molecules*; International Series of Monographs on Chemistry; Oxford: New York, 1989; Vol. *16*.
(15) Bartlett, R. J. Many-Body Perturbation Theory and Coupled Cluster Theory for Electron Correlation in Molecules. *Annu. Rev. Phys. Chem.* **1981**, *32*, 359−401.
(16) Jeziorski, B.; Moszynski, R.; Szalewicz, K. Perturbation Theory Approach to Intermolecular Potential Energy Surfaces of van der Waals Complexes. *Chem. Rev.* **1994**, *94*, 1887−1930.
(17) Szalewicz, K. Symmetry-adapted Perturbation Theory of Intermolecular Forces. *WIREs Comput. Mol. Sci.* **2012**, *2*, 254−272.
(18) Purvis, G. D.; Bartlett, R. J. A Full Coupled-cluster Singles and Doubles Model: The Inclusion of Disconnected Triples. *J. Chem. Phys.* **1982**, *76*, 1910−1918.
(19) Shavitt, I. In *Methods of Electronic Structure Theory*; Schaefer, H. F., Ed.; Plenum Press: New York, 1977; pp 189−275.
(20) Sherrill, C. D.; Schaefer, H. F. In *Adv. Quantum Chem.*; Löwdin, P.-O., Ed.; Academic Press: New York, 1999; Vol. *34*; pp 143−269.
(21) Boyd, D. B. *ACS Symp. Ser.* **2013**, *1122*, 221−273.
(22) NumPy Array Interface. https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.interface.html (accessed May 9th, 2018).
(23) Jones, E.; Oliphant, T.; Peterson, P. SciPy: Open source scientific tools for Python. http://www.scipy.org/ (accessed May 9th, 2018).
(24) Dask Development Team, Dask: Library for dynamic task scheduling. http://dask.pydata.org (accessed May 9th, 2018).
(25) Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I. J.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Józefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D. G.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P. A.; Vanhoucke, V.; Vasudevan, V.; Viégas, F. B.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv.org* **2016**, arXiv:1603.04467.
(26) Raffenetti, R. C. Pre-processing two-electron integrals for efficient utilization in many-electron self-consistent field calculations. *Chem. Phys. Lett.* **1973**, *20*, 335−338.
(27) Smith, D. G. A.; Støter, F.-R.; McGibbon, R. T.; Werner, N. Optimized Einsum: v1.0. *Zenodo* **2016**, DOI: 10.5281/zenodo.160842.
(28) Stanton, J. F.; Gauss, J.; Watts, J. D.; Bartlett, R. J. A direct product decomposition approach for symmetry exploitation in many-body methods. I. Energy calculations. *J. Chem. Phys.* **1991**, *94*, 4334−4345.
(29) Python Anaconda. https://www.anaconda.com (accessed May 9th, 2018).
(30) Fortenberry, R. C.; McDonald, A. R.; Shepherd, T. D.; Kennedy, M.; Sherrill, C. D. PSI4Education: Computational Chemistry Labs Using Free Software. *The Promise of Chemical Education: Addressing our Students Needs* **2015**, *1193*, 85−98.
(31) Pulay, P. Convergence acceleration of iterative sequences. The case of SCF iteration. *Chem. Phys. Lett.* **1980**, *73*, 393−398.
(32) Takeshita, T. Y.; de Jong, W. A.; Neuhauser, D.; Baer, R.; Rabani, E. Stochastic Formulation of the Resolution of Identity: Application to Second Order Møller-Plesset Perturbation Theory. *J. Chem. Theory Comput.* **2017**, *13*, 4605.
(33) Perez, F.; Granger, B. E. IPython: A System for Interactive Scientific Computing. *Comput. Sci. Eng.* **2007**, *9*, 21−29.
(34) Weiss, C. J. Scientific Computing for Chemists: An Undergraduate Course in Simulations, Data Processing, and Visualization. *J. Chem. Educ.* **2017**, *94*, 592−597.

DOI: 10.1021/acs.jctc.8b00286
J. Chem. Theory Comput. 2018, 14, 3504−3511