The Journal of Chemical Physics

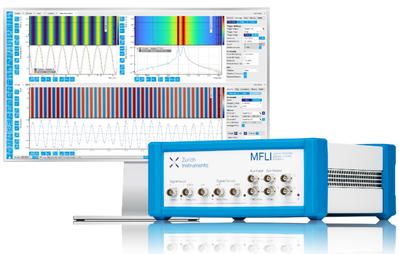
RESEARCH ARTICLE | OCTOBER 07 2024

ModelHamiltonian: A Python-scriptable library for generating 0-, 1-, and 2-electron integrals

Special Collection: Modular and Interoperable Software for Chemical Physics

Valerii Chuiko ; Addison D. S. Richards ; Gabriela Sánchez-Díaz ; Marco Martínez-González ; Wesley Sanchez ; Giovanni B. Da Rosa ; Michelle Richer ; Yilin Zhao ; William Adams ; Paul A. Johnson ; Farnaz Heidar-Zadeh ; Paul W. Ayers

J. Chem. Phys. 161, 132503 (2024)
https://doi.org/10.1063/5.0219015



The Journal of Chemical Physics | SOFTWARE | pubs.aip.org/aip/jcp

ModelHamiltonian: A Python-scriptable library for generating 0-, 1-, and 2-electron integrals

Cite as: J. Chem. Phys. 161, 132503 (2024); doi: 10.1063/5.0219015
Submitted: 14 May 2024 • Accepted: 13 August 2024 •
Published Online: 7 October 2024

Valerii Chuiko,[1] Addison D. S. Richards,[2] Gabriela Sánchez-Díaz,[1] Marco Martínez-González,[1] Wesley Sanchez,[1] Giovanni B. Da Rosa,[3] Michelle Richer,[1,4] Yilin Zhao,[1] William Adams,[1] Paul A. Johnson,[5] Farnaz Heidar-Zadeh,[4] and Paul W. Ayers[1,a)]

AFFILIATIONS

[1] Department of Chemistry and Chemical Biology, McMaster University, 1280 Main St. West, Hamilton, Ontario L8S 4M1, Canada
[2] Department of Physics and Astronomy, McMaster University, 1280 Main St. West, Hamilton, Ontario L8S 4M1, Canada
[3] Engineering School Télécom Paris, 19 Pl. Marguerite Perey, 91120 Palaiseau, France
[4] Department of Chemistry, Queen's University, 90 Bader Lane, Kingston, Ontario K7L 3N6, Canada
[5] Département de Chimie, Université Laval, Québec, Québec G1V 0A6, Canada

Note: This paper is part of the JCP Special Topic on Modular and Interoperable Software for Chemical Physics.

[a)] **Author to whom correspondence should be addressed:** ayers@mcmaster.ca

ABSTRACT

`ModelHamiltonian` is a free, open source, and cross-platform Python library designed to express model Hamiltonians, including spin-based Hamiltonians (Heisenberg and Ising models) and occupation-based Hamiltonians (Pariser–Parr–Pople, Hubbard, and Hückel models) in terms of 1- and 2-electron integrals, so that these systems can be easily treated by traditional quantum chemistry software programs. `ModelHamiltonian` was originally intended to facilitate the testing of new electronic structure methods using `HORTON` but emerged as a stand-alone research tool that we recognize has wide utility, even in an educational context. `ModelHamiltonian` is written in Python and adheres to modern principles of software development, including comprehensive documentation, extensive testing, continuous integration/delivery protocols, and package management. While we anticipate that most users will use `ModelHamiltonian` as a Python library, we include a graphical user interface so that models can be built without programming, based on connectivity/parameters inferred from, for example, a SMILES string. We also include an interface to ChatGPT so that users can specify a Hamiltonian in plain language (without learning `ModelHamiltonian`'s vocabulary and syntax). This article marks the official release of the `ModelHamiltonian` library, showcasing its functionality and scope.

I. WHAT IS `ModelHamiltonian`?

The `ModelHamiltonian` software library allows one to easily apply quantum chemistry methods to model Hamiltonians that are traditionally expressed using occupation number or spin operators. It achieves this by translating the models into 0-, 1-, and 2-electron integrals and then providing the user with those integrals in a useful format for interoperability with conventional quantum chemistry software packages (e.g., via `FCIDUMP` files)[1] and in-house programs (e.g., via `.npz` files). Currently, `ModelHamiltonian` natively supports the Pariser–Parr–Pople (PPP, also called extended Hubbard),[2,3] Hubbard,[4] Hückel,[5] XXZ Heisenberg,[6] Richardson–Gaudin,[7,8] and Ising[9] models. `ModelHamiltonian` also includes several utilities so that different notations and symmetries for integrals are interconvertible.

`ModelHamiltonian` is a component of the QC-Devs software ecosystem, which grew out of the `HORTON` project.[10,11] As a monolithic quantum chemistry package, `HORTON 2.x` failed to achieve our ambition to adhere to FAIR research software standards.[12] In particular, it was difficult for an outsider to use (much less extract and reuse) specific functionality of `HORTON`. Inspired by Ernest Davidson's `MELD` package,[13] we decomposed `HORTON` into specific packages, each with well-defined scope and a clear and intuitive API. `ModelHamiltonian` is designed to work seamlessly within this ecosystem, which includes the following:

- `IOData` for file parsing, writing, and conversion.[14]

- `Gbasis` and `BFit` for manipulating, evaluating, and fitting to Gaussian basis functions.[15–17]
- `FanPy` and `PyCI` for *ab initio* quantum chemistry calculations.[18,19]
- Various utility modules, especially related to numerical integration/differentiation, linear algebra, and post-processing quantum chemistry calculations.[20–23]

On its own and through `IOData`, `ModelHamiltonian` is designed to be interoperable with third-party packages. For example, we routinely use `ModelHamiltonian` in conjunction with `PySCF`.[24,25]

The original motivation of `ModelHamiltonian` was assessing new approaches for selected configuration interaction and new (nonlinear) wavefunction ansatze.[26–31] For example, when testing new wavefunction methods, it is common to use small hydrogen clusters, which, depending on the relative arrangement and relative distance between atoms, allow one to mimic the range and types of correlation that are seen in PPP/Hubbard models.[32–51] The decision to explore hydrogen clusters seems to be less motivated by any physical or chemical relevance to these systems than by the ease of treating such systems in traditional electronic structure modeling packages, where the typical input is atoms' positions and their nuclear charges (atomic numbers). While hydrogen clusters can mimic some of the characteristics of PPP/Hubbard models, it is much more difficult, technically, to impose periodic boundary conditions on atomic/molecular clusters. By contrast, `ModelHamiltonian` provides native support for periodic boundary conditions, which are easy to implement for lattice models. As our interests grew to include machine-learning approaches to electronic structure[52–56] theory,[57] it became important to generate vast quantities of benchmark and training data. At that time, we could not find any Python library that could automatically generate the 1- and 2-electron integrals needed as input for quantum chemistry packages for the variety of different occupation- and spin-based Hamiltonians we were interested in. Accordingly, we took the in-house tool we had used to treat model Hamiltonians in the past and converted it into the scriptable `ModelHamiltonian` library.

In Secs. II and III, we present the structure of `ModelHamiltonian` and present examples of its use. Then, *in lieu* of a conventional summary, we conclude by answering some frequently asked questions.

II. STRUCTURE OF ModelHamiltonian

A. Structure of python library

`ModelHamiltonian` is written in Python 3, leveraging `NumPy`[58] and `SciPy`[59] libraries to achieve speed and memory efficiency. There are three main object-oriented components in `ModelHamiltonian`.

1. Hamiltonian API class

All supported Hamiltonians are based on the `HamiltonianAPI` class. Within this class, Hamiltonians are defined in second-quantized form using physics notation,[60]

$$\hat{H} = \sum_{pq} h_{pq} a_p^\dagger a_q + \frac{1}{2}\sum_{pqrs} g_{pqrs} a_p^\dagger a_q^\dagger a_s a_r, \tag{1}$$

where

$$h_{pq} = \int \phi_p^*(\mathbf{r})\left(-\frac{\nabla^2}{2} - \sum_{n=1}^{N_{\text{atoms}}} \frac{Z_n}{|\mathbf{r}-\mathbf{R}_n|}\right)\phi_q(\mathbf{r})\, d\mathbf{r}, \tag{2a}$$

$$g_{pqrs} = \iint \phi_p^*(\mathbf{r}_1)\phi_q^*(\mathbf{r}_2)\frac{1}{|\mathbf{r}_1-\mathbf{r}_2|}\phi_r(\mathbf{r}_1)\phi_s(\mathbf{r}_2)\, d\mathbf{r}_1 d\mathbf{r}_2, \tag{2b}$$

and $a_p^\dagger$ and a_p denote the fermion creation and annihilation operators for an electron in the pth spin-orbital. Where it is necessary to denote the spin explicitly, we write $a_{p\alpha}^\dagger$ or $a_{q\beta}$.

All Hamiltonians are stored in the spin-orbital basis, but, when spin is not specified, we assume that the integrals preserve spin-symmetry. This leads us to the first general property of any Hamiltonian: *interconversion between spin- and spatial orbitals*. In particular, we use the following mapping:

$$h_{pq} = h_{p_\alpha q_\alpha} = h_{p_\beta q_\beta}, \tag{3a}$$

$$g_{pqrs} = \frac{g_{p_\alpha q_\alpha r_\alpha s_\alpha} + g_{p_\alpha q_\beta r_\alpha s_\beta}}{2}, \tag{3b}$$

and assume, using spin-symmetry, that

$$\begin{aligned} g_{p_\alpha q_\alpha r_\alpha s_\alpha} &= g_{p_\beta q_\beta r_\beta s_\beta}, \\ g_{p_\alpha q_\beta r_\alpha s_\beta} &= g_{p_\beta q_\alpha r_\beta s_\alpha}. \end{aligned} \tag{4}$$

It is important to note that the common assumption of highly symmetric integrals, such as those with eightfold symmetry, is not compatible with all model Hamiltonians. That is why in addition to interconversion between integrals expressed in spin- and spatial orbitals, the `HamiltonianAPI` class supports *conversion between two-, four-, and eightfold symmetries*. In particular,

Twofold symmetry:

$$h_{i,j} = h_{j,i}, \tag{5a}$$

$$g_{i,j,k,l} = g_{k,l,i,j}. \tag{5b}$$

Fourfold symmetry:

$$g_{i,j,k,l} = g_{k,l,i,j} = g_{j,i,l,k} = g_{l,k,j,i}. \tag{5c}$$

Eightfold symmetry:

$$g_{i,j,k,l} = g_{k,l,i,j} = g_{j,i,l,k} = g_{l,k,j,i} \tag{5d}$$

$$= g_{k,j,i,l} = g_{i,l,k,j} = g_{l,i,j,k} = g_{j,k,l,i}. \tag{5e}$$

Zero-, one-, and two-electron integrals are generated by calling the following methods:

```
# generate zero energy shift
generate_zero_body_integral()
# generate one-electron integrals
generate_one_body_integral(dense: bool,
                           basis: str)
# generate two-electron integrals
generate_two_body_integral(sym: int,
                           basis: str,
                           dense: bool)
```

While it is somewhat unconventional, we refer to constant shifts, related to the conventional choice for the zero of energy, as zero-body integrals. Once generated, one- and two-electron integrals can be returned as a `numpy.ndarray`[58] (`dense = True`) or `scipy.sparse.csr_matrix`[59] (`dense = False`). As `scipy.sparse.csr_matrix`[59] can store only two-dimensional arrays, sparse two-electron integrals are expressed using the two-to-four index mapping, $\{i,j\} \leftrightarrow \{p,q,r,s\}$,

$$i = p \cdot n + q, \tag{6a}$$

$$j = q \cdot n + r. \tag{6b}$$

Once generated, Hamiltonians can be saved as either `FCIDUMP`[1] or `.npz`[58] files. Users can provide either a `TextIO` object or a `str` corresponding to a filename,

```
# save to FCIDUMP file
save_fcidump(f: Union[TextIO, str],
             nelec=0,
             spinpol=0)

# save to NPZ file
savez(f: Union[TextIO, str])
```

2. Pariser–Parr–Pople class

The Pariser–Parr–Pople[2,3] (PPP) subclass of `Hamiltonian` stores the zero-, one-, and two-electron integrals for Hamiltonians of the form

$$\begin{aligned}\hat{H}_{\mathrm{PPP}} &= \sum_{pq} h_{pq} a_p^\dagger a_q + \sum_p U_p \hat{n}_{p\alpha}\hat{n}_{p\beta} \\ &\quad + \frac{1}{2}\sum_{p\neq q} \gamma_{pq}\left(\hat{n}_{p\alpha}+\hat{n}_{p\beta}-Q_p\right)\left(\hat{n}_{q\alpha}+\hat{n}_{q\beta}-Q_q\right),\end{aligned} \tag{7}$$

where

$$\hat{n}_{p\sigma} = a_{p\sigma}^\dagger a_{p\sigma}, \qquad \sigma \in \{\alpha,\beta\}, \tag{8}$$

U_p is the on-site Coulomb repulsion, Q_p is the background (excess) charge of the site, and γ_{pq} is the inter-site Coulomb repulsion. In our notation, α corresponds to spin-up electrons and β corresponds to spin-down electrons.

Two Hamiltonians inherit the PPP class: the Hubbard Hamiltonian and the Hückel Hamiltonian.[4] The Hubbard Hamiltonian is obtained from the PPP Hamiltonian [Eq. (7)] by setting inter-site Coulomb interactions to zero ($\gamma_{pq} = 0$),

$$\hat{H}_{\mathrm{Hubbard}} = \sum_{pq} h_{pq} a_p^\dagger a_q + \sum_p U_p \hat{n}_{p\alpha}\hat{n}_{p\beta}. \tag{9}$$

The Hückel Hamiltonian[5] inherits the Hubbard model and corresponds to zero on-site Coulomb repulsion, $U_p = 0$,

$$\hat{H}_{\mathrm{H\ddot{u}ckel}} = \sum_{pq} h_{pq} a_p^\dagger a_q. \tag{10}$$

3. XXZ Heisenberg class

Sometimes, it is sensible to assume that sites in a molecule or material are associated with spins; this typically occurs when the number of electrons on a site is basically fixed, but the spin of the electron on the site is variable. Spin-operators can be cast in terms of fermion creation/annihilation operators in several different ways. While the most familiar approach is probably the Jordan–Wigner transformation,[61–63] there are many different choices.[64–66] It is convenient to write spins as pairs of electrons,[29,67–69]

$$S_p^+ = a_{p\alpha}^\dagger a_{p\beta}^\dagger, \tag{11a}$$

$$S_p^- = a_{p\beta} a_{p\alpha}, \tag{11b}$$

$$S_p^Z = \frac{1}{2}\left(a_{p\alpha}^\dagger a_{p\alpha} + a_{p\beta}^\dagger a_{p\beta} - 1\right). \tag{11c}$$

The other Cartesian-spin operators are

$$S_p^X = \frac{1}{2}\left(S_p^+ + S_p^-\right), \tag{12a}$$

$$S_p^Y = \frac{1}{2i}\left(S_p^+ - S_p^-\right). \tag{12b}$$

This definition for $\{S_p^\pm, S_p^Z\}$ maps spin Hamiltonians to seniority-zero fermionic Hamiltonians, which is convenient given the recent interest in practical (and impractical) seniority-zero methods[27,28,30,31,43,44,70–92] and associated software packages.[18,19,24,25,93,94] The other conventional choice (e.g., using $S_p^+ = a_{p\alpha}^\dagger a_{p\beta}$) maps spin Hamiltonians to Hamiltonians with maximum seniority, which we find less convenient.

The XXZ Heisenberg model is more amenable to quantum chemistry software since the XYZ Heisenberg model[6] leads to non-number-conserving terms (in particular, terms such as $a_{p\beta}a_{q\beta}a_{p\alpha}a_{q\alpha}$ and $a_{p\alpha}^\dagger a_{q\alpha}^\dagger a_{p\beta}^\dagger a_{q\beta}^\dagger$) when spins are mapped onto the fermion creator–annihilator operators using Eq. (11). The specific model we consider is

$$\hat{H}_{XXZ} = \sum_p \mu_p^Z S_p^Z + \sum_{pq}\left[J_{pq}^{\mathrm{ax}} S_p^Z S_q^Z + J_{pq}^{\mathrm{eq}}\left(S_p^X S_q^X + S_p^Y S_q^Y\right)\right] \tag{13}$$

or, equivalently,

$$\hat{H}_{XXZ} = \sum_p \mu_p^Z S_p^Z + \sum_{pq} J_{pq}^{\mathrm{ax}} S_p^Z S_q^Z + \frac{1}{2}\sum_{pq} J_{pq}^{\mathrm{eq}}\left(S_p^+ S_q^- + S_p^- S_q^+\right). \tag{14}$$

The topology of the Heisenberg model is encoded by setting $J_{pq}^{\mathrm{ax}} = J_{pq}^{\mathrm{eq}} = 0$ for sites that are not adjacent/connected. Using the commutation relations, we can rewrite Eq. (14) in normal order as

$$\hat{H}_{XXZ} = \sum_p \left(\mu_p^Z - J_{pp}^{\mathrm{eq}}\right) S_p^Z + \sum_{pq} J_{pq}^{\mathrm{ax}} S_p^Z S_q^Z + \sum_{pq} J_{pq}^{\mathrm{eq}} S_p^+ S_q^- \tag{15}$$

and then map it to fermion creation/annihilation operators using Eq. (11), obtaining

$$\begin{aligned}\hat{H}_{XXZ} &= \sum_p \left(\mu_p^z - J_{pp}^{eq}\right) a_p^+ a_p \\ &\quad + \frac{1}{4}\sum_{pq} J_{pq}^{ax}\left(\hat{n}_{p\alpha}+\hat{n}_{p\beta}-1\right)\left(\hat{n}_{q\alpha}+\hat{n}_{q\beta}-1\right) \\ &\quad + \sum_{pq} J_{pq}^{eq} a_{p\alpha}^+ a_{p\beta}^+ a_{q\beta} a_{q\alpha}.\end{aligned} \tag{16}$$

Note that this mapping from spins to fermions results in a seniority-zero Hamiltonian.[46] Only the seniority-zero eigenspace of this Hamiltonian represents solutions to the XXZ Heisenberg model. Nonetheless, this transformation is interesting because, as noted by Scuseria's group, the fermion "dual" to the Heisenberg Hamiltonian is frequently more amenable to computation.[95]

This mapping of spin-Hamiltonians to fermionic seniority-zero Hamiltonians makes it easier to apply (un)conventional electronic-structure methods to spin Hamiltonians, although one must be careful to interpret only the seniority-zero solutions. (Higher-seniority solutions can be practically excluded by (manually) specifying the non-seniority-zero terms in the Hamiltonian to be large and positive, thereby ensuring that the low-energy eigenstates have seniority zero.) Unless one is using a code that is adapted to seniority-zero systems, however, this approach sacrifices computational cost for human convenience.

Two classes inherit the XXZ Heisenberg model.[6] When only axial part of exchange coupling is present ($J_{pq}^{eq} = 0$), one obtains the Ising model,[9]

$$\hat{H}_{\text{Ising}} = \sum_p \mu_p^Z S_p^Z + \sum_{pq} J_{pq}^{\text{ax}} S_p^Z S_q^Z. \tag{17}$$

If only the equatorial part of exchange coupling is present, $J_{pq}^{ax} = 0$, we get a generalization of the Richardson–Gaudin model,[7,8]

$$\hat{H}_{\text{gen-RG}} = \sum_p \left(\mu_p^Z - J_{pp}^{\text{eq}}\right) S_p^Z + \sum_{pq} J_{pq}^{\text{eq}} S_p^+ S_q^-. \tag{18}$$

This Hamiltonian is integrable, provided that the equatorial couplings either are completely degenerate $J_{pq}^{\text{eq}} = J^{\text{eq}}$ or can be factorized $J_{pq}^{\text{eq}} = \sqrt{J_{pp}^{\text{eq}} J_{qq}^{\text{eq}}}$.

The preceding discussion is encapsulated in the class inheritance diagram (Fig. 1).

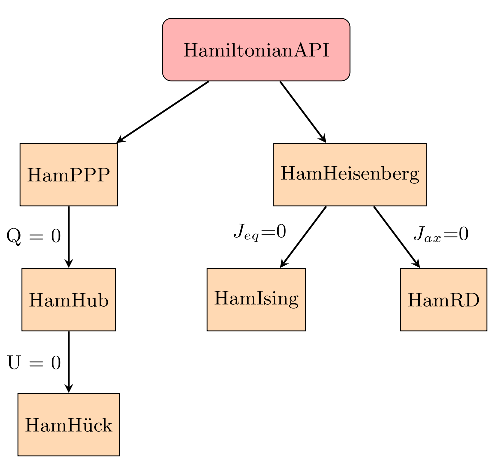


FIG. 1. Overview of the structure of `ModelHamiltonian`.

III. USING `ModelHamiltonian`

A. Documentation and tutorials

`ModelHamiltonian`'s website, modelh.qcdevs.org, provides thorough up-to-date documentation, including installation instructions, detailed tutorials (complete with their corresponding Jupyter Notebooks), and API documentation.

B. Basic usage (Python)

`ModelHamiltonian` can be used directly in Python scripts and programs. A minimal script requires (1) specifying connectivity, (2) specifying the Hamiltonian type (and its parameters), and (3) building the integrals.

1. Specifying connectivity

`ModelHamiltonian` supports two ways to specify the connectivity of an occupation-number-based Hamiltonian. First, one can directly provide an adjacency matrix for the sites. Alternatively, one can specify atom types and their connectivity,

```python
import numpy as np
import moha

# defining 4 site Huckel model
# with periodic boundary condtions
# using adjacency matrix:
connectivity_1 = np.array([[0, 1, 0, 1],
                           [1, 0, 1, 0],
                           [0, 1, 0, 1],
                           [1, 0, 1, 0]])
# second option
# defining connectivity by
# specifying atom types:
connectivity_2 = [("C1", "C2", 1),
                  ("C2", "C3", 1),
                  ("C3", "C4", 1),
                  ("C4", "C1", 1)]
```

2. Specifying a Hamiltonian

Given the connectivity, the user defines the Hamiltonian by initializing its class. For example, the Hückel model can be defined using the `HamHuck` class. One can always specify the parameters for the model (e.g., α and β for the site energy and the hopping/inter-site energy). By default, $\alpha = -0.414$ a.u. and $\beta = -0.533$ a.u., corresponding to electrons in a carbon 2p orbital and a π-bonding interaction between neighboring sites,[96]

```python
# Creating the Huckel Model using HamHuck class
ham = HamHub(connectivity_2,
             alpha=-0.414, beta=-0.0533)
```

Similarly, one can create an instance of the Hubbard Hamiltonian the `HamHub` class; the potential on each site `u_onsite` needs to be specified,

```
# define Hubbard model with alpha=0,
# beta=-1, and U_p = 1 for all p
hubbard = HamHub(connectivity_1,
                 alpha=0,
                 beta=-1,
                 u_onsite=np.array(
                         [1, 1, 1, 1]
                        )
                )
```

Working with spin-based Hamiltonians is similar, but the `XXZ Heisenberg` class supports only the adjacency matrix to specify connectivity and the site energy and axial exchange coupling must be specified. For example, the following code snippet builds a 4-site XXZ Heisenberg chain with periodic boundary conditions and no external field:

```
# import libraries
import numpy as np
from moha import HamHeisenberg

# Defining the Hamiltonian
# 4 sites 1d chaid
# with periodic boundary condition
# Define the Hamiltonian parameters
J_eq = 0.5
J_ax = 1
mu = 0
# Define the Hamiltonian
ham = HamHeisenberg(connectivity_1,
                    J_eq=J_eq,
                    J_ax=J_ax,
                    mu=mu)
```

3. Building the Hamiltonian

After creating the desired Hamiltonian, one generates 0-, 1-, and 2-electron integrals by calling the corresponding methods. For example, the following code snippet returns one-electron integrals as a dense NumPy array in the spatial basis and two-electron integrals in the spin-orbital basis as a NumPy array (with eightfold symmetry):

```
h1 = ham.generate_one_body_integral(
               dense=True,
               basis='spatial basis'
               )
h2 = ham.generate_two_body_integral(
               dense=True,
               basis='spinorbital basis',
               sym=8
               )
```

C. Alternative input methods

We anticipate that users who are comfortable with object-oriented programming in Python will work directly with `ModelHamiltonian`, as it allows full control. However, users have different levels of familiarity with Python programming, and for using `ModelHamiltonian` in the context of an undergraduate quantum chemistry course (cf. qchem.qc-edu.org), we find that alternative input styles are useful.

1. GPT

We introduce a new strategy to enhance usability by integrating ChatGPT[97] functionality into `ModelHamiltonian`. The idea is that the user can specify their Hamiltonian in plain language and avoid learning the vocabulary and syntax of `ModelHamiltonian`. We believe that this makes it easier to use `ModelHamiltonian` and helps users build intuition and understanding about the key ingredients in model Hamiltonians.

a. Setup and configuration. To take advantage of the ChatGPT[97] module, the `openai` subversion of `ModelHamiltonian` should be installed and an OpenAI API key needs to be provided. This process is discussed in detail in a tutorial on the `ModelHamiltonian` website.

b. Usage and flexibility. To ensure the accuracy of the Hamiltonians generated by ChatGPT,[97] certain restrictions were imposed. When defining the system, all model-specific parameters are specified as numerical values, which are then mapped onto the connectivity. This helps ensure that the generated Hamiltonian accurately reflects the intended system and maintains consistency throughout the modeling process. In the prompt, the user has control over the type of generated Hamiltonian, the number of electrons and orbitals, and the type of boundary conditions. Users can also control the symmetry, output style, and output format/location of the integrals.

As a simple example of the power and flexibility of natural language input, we regenerated the Hückel model embedded in the preceding Hubbard-model and save the integrals in an FCIDUMP file named `huckel_1`,

```
from moha.gpt import generate_ham

# defining Huckel model
huck = generate_ham(
"Generate a simple 1d Huckel model\
with periodic boundary condition and t=-0.053.\
Set energy of orbital equal to -0.414.\
System has 4 sites and 4 orbitals.\
Return ouput in the spatialbasis.\
Save it to the fcidump file called huckel_1d"
)
```

In this example, the natural language description is longer than one would have with direct invocation of `ModelHamiltonian`'s functionality, so the only gain is that the user does not need to read the code's documentation. Sometimes, the plain-text description can be much shorter than explicitly invoking `ModelHamiltonian` because of the contextual awareness of ChatGPT.[97] Below, we generate a two-dimensional Richardson–Gaudin model with 6 electrons and periodic (toroidal) boundary conditions. We can explicitly construct the Hamiltonian using the following code:

```python
from moha.hamiltonians import HamRG

Lx, Ly = 3, 2
nsites = Lx * Ly
adjacency = np.zeros((nsites, nsites))
for n in range(nsites):
    nx = n % Lx  # x index
    ny = n // Lx  # y index
    ndx = (nx + 1) % Lx  # x shift
    ndy = (ny + 1) % Ly  # y shift

    # add x neighbours to connectivity
    dn = ndx + Lx * ny
    adjacency[n, dn] = 1
    # add y neighbours to connectivity
    dn = nx + Lx * ndy
    adjacency[n, dn] = 1
adjacency += adjacency.T

rg = HamRg(
    connectivity = adjacency,
    mu = 0,
    J_eq = 0.5
)
# generating integrals
e0 = ham.generate_zero_body_integral()
h1 = ham.generate_one_body_integral(
            dense=True,
            basis='spatial basis'
            )
h2 = ham.generate_two_body_integral(
            dense=True,
            basis='spatial basis',
            sym=4)

# saving integrals
rg.savez("RG_4")
```

The same functionality can be accessed with the natural language input,

```python
rg = generate_ham(
"Generate a 2d Richardson-Gaudin model\
that has 6 electrons. \
It should have a 3x2 size \
with periodic boundary condition. \
Coupling J is 0.5.\
Return ouput in the spatial basis.\
Assume 4-fold symmetry.\
Save result as npz file RG_4.")
```

ChatGPT infers from the specification of the Richardson–Gaudin model that $J_{ax} = 0$ and, even more impressively, correctly generates site connectivity from the specification of the boundary conditions.

We believe that ChatGPT functional is incredibly promising as it allows users to avoid learning the detailed vocabulary and syntax of a software package. We note, however, that the reliability of ChatGPT was enhanced by the fact `ModelHamiltonian` is a well-structured and well-documented code base with a clear API.

2. TOML

For users who prefer to interact with `ModelHamiltonian` through an input file, we offer the option to specify model parameters using TOML[98] files. Although this format restricts users to the same functionality as ChatGPT (Sec. III C 1)-based input, it requires a strict keyword format. In a TOML[98] file, users specify program options within sections such as *control*, *system*, and *model*. The following TOML file generates a 1D Hubbard chain:

```toml
[control]
prefix = "hubbard"
outdir = "./fcidump"
save_integrals = true
integral_format = "fcidump"

[system]
moltype = "1d"
bc = "open"
norb = 6
nelec = 6

[model]
hamiltonian = "Hubbard"
basis = "spatial basis"
beta = -1.0
u_onsite = 1.0
```

In the *control* section, users can specify parameters such as the prefix for file names, the output directory, and whether to save integrals (`true` or `false`). The *system* section allows users to define properties of the quantum system, such as the dimensionality of the system (`moltype = {"1d", "2d"}`), boundary conditions (`bc = {"open", "periodic"}`), number of orbitals, and number of electrons. Similarly, the *model* section enables users to specify characteristics of the model Hamiltonian, including the type of Hamiltonian (`hamiltonian = {"PPP", "Hubbard", "Huckel", "Heisenberg", "Ising", "RG"}`), basis type (`basis = {"spatial basis", "spinorbital basis"}`), and other relevant parameters for the specified model. The parameters' names are fully consistent with the Python-based API. More examples of TOML files can be found on the GitHub repository under the folder `/examples/toml`.

After generating the TOML file, the user can get generate the Hamiltonian via Python IDE as follows:

```python
from moha.toml_tools import from_toml
ham = from_toml("my_toml_file.toml")
```

3. Graphical user interface (GUI)

Building a Hamiltonian for complex systems can be challenging even for users who are capable programmers. Accordingly, we provide a GUI, wherein users can specify parameters using keywords,

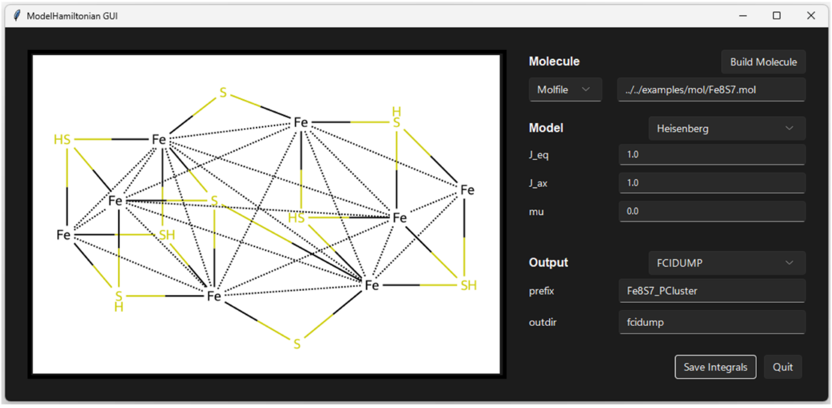


FIG. 2. GUI of ModelHamiltonian.

mirroring the structure of and options in TOML files (Sec. III C 2). To showcase this approach, we construct a Heisenberg model for a p-cluster system, a real model Hamiltonian studied in Ref. 99 (Fig. 2).

Figure 2 shows a snapshot of the GUI. In addition to the TOML-input, users have the option to obtain connectivity information directly from a SMILES[100] string or a MOL file.[101] However, when using a MOL file, only the information from the symbolic bonds is used for building the adjacency matrix.

`ModelHamiltonian`'s GUI is invoked from the command line:

```
python ModelHamiltonian/moha/gui/moha_gui.py.
```

D. Working with other programs

We built `ModelHamiltonian` to be interoperable with other programs so that users can integrate it into their normal workflow. For example, using the FCIDUMP file, one can use `ModelHamiltonian` with packages and libraries, including `PySCF`,[24] `GBasis`,[15] and `IOData`.[14] Through `IOData`'s file-conversion capabilities, other packages can be used.

When using `ModelHamiltonian`, the most difficult part of the system specification is typically the sites' connectivity. In the following example, we show how one can infer adjacency matrix from a SMILES string[100] using `RDKit`.[102] Users can use the generated one- and two-electron integrals directly as input to `PySCF`, which can then find the ground-state energy and wavefunction. Note that when using `PySCF`, one needs to convert the 2-electron integrals from physicists' (default) to chemists' notation: $g^{ph}_{pqrs} = g^{ch}_{prqs}$.

The following code generates a Hubbard model that is appropriate for the naphthalene molecule, passes it to `PySCF`, and determines the ground-state energy and wavefunction using a full configuration interaction calculation:

```
from rdkit.Chem import MolFromSmiles, rdmolops
from pyscf import fci

# generate a molecule from SMILES
mol = MolFromSmiles('C1=CC=C2C=CC=CC2=C1')
# generate the connectivity matrix
connectivity = rdmolops.GetAdjacencyMatrix(mol)
# generate Hubbard model
# setting on-site repulsion to
# 10.84 eV for each atom.
U = 10.84 * np.ones(connectivity.shape[0])
hubbard = HamHub(connectivity,
                 alpha=0, beta=-1,
                 u_onsite=U)
e0_hub = hubbard.generate_zero_body_integral()
h1_hub = hubbard.generate_one_body_integral(
            dense=True,
            basis='spatial basis')
h2_hub = hubbard.generate_two_body_integral(
            dense=True,
            basis='spatial basis',
            sym=8)

# convert h2 to chemists notation
h2_ch = np.transpose(h2_hub, (0, 2, 1, 3))

# Calculate eigenvalues and ground state
e, ci = fci.direct_spin0.kernel(
            h1_hub,
            h2_ch,
            10,# number of orbitals
            10# number of electrons
            )
```

Alternatively, one can use the connectivity matrix to construct and solve an XXZ Heisenberg model,

```python
# import libraries
from pyscf import fci
# Define the Hamiltonian
ham = HamHeisenberg(connectivity,
                    J_eq=0.5,
                    J_ax=1,
                    mu=0)
e0 = ham.generate_zero_body_integral()
h1 = ham.generate_one_body_integral(
            dense=True,
            basis='spatial basis'
            )
h2 = ham.generate_two_body_integral(
            dense=True,
            basis='spatial basis',
            sym=4)

# convert h2 to chemists notation
h2_ch = np.transpose(h2, (0, 2, 1, 3))

# Calculate eigenvalues and ground state
e, ci = fci.direct_spin0.kernel(
            h1,
            h2_ch,
            4,# number of orbitals
            4# number of electrons
            )
```

If one would like to save this Hamiltonian for future use, it can be saved as an NPZ file or an FCIDUMP file, as shown in the following:

```python
# saving the integrals as a npz file
ham.savez("XXZ_4.npz")
# saving the integrals as fcidump file
ham.save_fcidump(
        open('XXZ_4.fcidump', 'w'),
        nelec=4
        )
```

IV. FREQUENTLY ASKED QUESTIONS

A. Who/what is ModelHamiltonian for?

ModelHamiltonian was initially designed for researchers in quantum chemistry, particularly those focusing on strongly correlated systems. Its primary aim is to make it easier to benchmark, and perform comparative studies of, quantum chemistry methods. While a few quantum chemistry packages support some model Hamiltonians,[24,25,103] we do not know of any package that supports a comparable range of model Hamiltonians.

In addition, we also find `ModelHamiltonian` useful for generating training data for machine-learning research in quantum chemistry and solid-state physics. For such applications, model Hamiltonians are more useful than real molecular systems because they give scientists fine-grained control over the strength and nature of electron correlation.

B. What is the future direction of ModelHamiltonian?

We will make `ModelHamiltonian` even more flexible by supporting multiple atoms and bond types, including libraries of default parameters appropriate for molecules and solids. We are also adding Hamiltonians that are both occupation- and spin-based, such as the t-J-U-V[104,105] and t-J models.[106–108] Finally, we will increase the usability of `ModelHamiltonian` by allowing Hamiltonians to be specified and generated directly from the command line and by improving the graphical user interface.

C. How do I install `ModelHamiltonian`?

Installation of `ModelHamiltonian` requires the following programs and libraries to be installed on your system:

- Git,
- Python 3,
- NumPy, and
- SciPy.

To install `ModelHamiltonian`, you must clone the Git repository containing the code and then install the Python module,

```bash
# Clone the Git repository
git clone https://github.com/theochem/ModelHamiltonian
# Enter the ModelHamiltonian directory
cd ModelHamiltonian
# Install the Python package using pip
python -m pip install .
```

D. How can I contribute to `ModelHamiltonian`?

We welcome and support new contributions in accordance with our Code of Conduct and Contributing Guidelines. For the most up-to-date instructions on how to contribute, please refer to the `ModelHamiltonian` website (modelh.qcdevs.org) or the README file on the GitHub page.

ACKNOWLEDGMENTS

The authors acknowledge support of their research from the Natural Sciences and Engineering Research Council of Canada and the Digital Research Alliance of Canada. In addition, PWA acknowledges support from the Canada Research Chairs and ADSR appreciates the support of Erik Sorensen (NSERC Discovery Grant No. RGPIN-2017-05759).

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Valerii Chuiko: Conceptualization (equal); Methodology (equal); Software (lead); Supervision (equal); Writing – original draft (lead); Writing – review & editing (equal). **Addison D. S. Richards**: Methodology (equal); Software (equal); Writing – review & editing (supporting). **Gabriela Sánchez-Díaz**: Software (equal). **Marco Martínez-González**: Software (equal). **Wesley Sanchez**: Software (equal). **Giovanni B. Da Rosa**: Software (equal). **Michelle Richer**: Software (equal). **Yilin Zhao**: Conceptualization (equal); Software (supporting). **William Adams**: Software (supporting). **Paul A. Johnson**: Conceptualization (supporting); Methodology (supporting); Writing – review & editing (supporting). **Farnaz Heidar-Zadeh**: Conceptualization (supporting); Funding acquisition (equal); Methodology (equal); Project administration (supporting); Resources (supporting); Software (supporting); Writing – review & editing (supporting). **Paul W. Ayers**: Conceptualization (lead); Formal analysis (lead); Funding acquisition (equal); Methodology (lead); Software (equal); Supervision (lead); Writing – original draft (equal); Writing – review & editing (equal).

DATA AVAILABILITY

Examples from the paper (and more) are available online at modelh.qcdevs.org. The ModelHamiltonian code is hosted on GitHub at https://github.com/theochem/ModelHamiltonian.

APPENDIX: INTEGRATION OF ChatGPT FUNCTIONALITY

In this appendix, we explain how we integrate ChatGPT with `ModelHamiltonian`.

1. Core functionality definition: We established the core functionality of the `ModelHamiltonian` package, which revolves around generating Hamiltonians based on user-provided parameters specific to the selected type of Hamiltonian.
2. Connectivity definition: For each selected Hamiltonian, we defined the connectivity between elements, taking into account the lattice type (1D or 2D) and boundary conditions (open or closed).
3. Output format selection: Upon specifying the Hamiltonian, users are prompted to select the desired output format, including the matrix type (sparse or dense) and whether to save integrals. If saving is preferred, users can specify the output format (FCIDUMP or NPZ).
4. Function association with keywords: A set of functions was developed to execute each action, such as generating Hamiltonians, defining connectivity, and selecting output formats. These functions are associated with specific keywords, mirroring those found in the TOML file.
5. Mapping natural language prompts: Leveraging the OpenAI API, natural language prompts from users are mapped onto a predefined set of keywords that trigger the corresponding functions defined in the previous step. This mapping ensures accurate interpretation and execution of user commands within the `ModelHamiltonian` package.

To mitigate the risk of hallucinations that may occur when using ChatGPT to map prompts onto keywords, we adopted a function calling approach. By allowing users to describe their Hamiltonian to the Assistant's API, we intelligently return the functions (keywords) that need to be called along with their arguments (allowed values). This approach significantly reduces the likelihood of erroneous interpretations and enhances the reliability and usability of the ChatGPT integration in the `ModelHamiltonian` package.

REFERENCES

[1] P. J. Knowles and N. C. Handy, "A determinant based full configuration interaction program," Comput. Phys. Commun. **54**, 75–83 (1989).

[2] J. A. Pople, "Electron interaction in unsaturated hydrocarbons," Trans. Faraday Soc. **49**, 1375–1385 (1953).

[3] R. Pariser and R. G. Parr, "A semi-empirical theory of the electronic spectra and electronic structure of complex unsaturated molecules. II," J. Chem. Phys. **21**, 767–776 (1953).

[4] J. Hubbard and B. Hilton Flowers, "Electron correlations in narrow energy bands," Proc. R. Soc. London, Ser. A **276**, 238–257 (1963).

[5] E. Hückel, "Quantentheoretische beiträge zum benzolproblem: I. Die elektronenkonfiguration des benzols und verwandter verbindungen," Z. Phys. **70**, 204–286 (1931).

[6] W. Heisenberg, "Zur theorie des ferromagnetismus," Z. Phys. **49**, 619–636 (1928).

[7] R. W. Richardson, "Exact eigenstates of the pairing-force Hamiltonian. II," J. Math. Phys. **6**, 1034–1051 (1965).

[8] M. Gaudin, *La fonction d'onde de Bethe* (Elsevier Masson, Barcelona, Spain, 1983).

[9] E. Ising, "Beitrag zur theorie des ferromagnetismus," Z. Phys. **31**, 253–258 (1925).

[10] T. Verstraelen, P. Tecmer, F. Heidar-Zadeh, C. E. González-Espinoza, M. Chan, T. D. Kim, K. Boguslawski, S. Fias, S. Vandenbrande, D. Berrocal, and P. W. Ayers (2017). "HORTON 2.1.1," Github. http://theochem.github.io/horton/2.1.1/.

[11] M. Chan, T. Verstraelen, A. Tehrani, M. Richer, X. D. Yang, T. D. Kim, E. Vöhringer-Martinez, F. Heidar-Zadeh, and P. W. Ayers, "The tale of HORTON: Lessons learned in a decade of scientific software development," J. Chem. Phys. **160**, 162501 (2024).

[12] M. Barker, N. P. Chue Hong, D. S. Katz, A.-L. Lamprecht, C. Martinez-Ortiz, F. Psomopoulos, J. Harrow, L. J. Castro, M. Gruenpeter, P. A. Martinez, and T. Honeyman, "Introducing the FAIR Principles for research software," Sci. Data **9**, 622 (2022).

[13] E. R. Davidson, "MELD: A many electron description," in *MOTECC-94: Methods and Techniques in Computational Chemistry*, edited by E. Clementi (STEF, Calgliari, Italy, 1993), Vol. B, pp. 209–274.

[14] T. Verstraelen, W. Adams, L. Pujal, A. Tehrani, B. D. Kelly, L. Macaya, F. Meng, M. Richer, R. Hernández-Esparza, X. D. Yang *et al.*, "IOData: A Python library for reading, writing, and converting computational chemistry file formats and generating input files," J. Comput. Chem. **42**, 458–464 (2021).

[15] T. D. Kim, L. Pujal, M. Richer, M. van Zyl, M. Martínez-González, A. Tehrani, V. Chuiko, G. Sánchez-Díaz, W. Sanchez, W. Adams, X. Huang, B. D. Kelly, E. Vöhringer-Martinez, T. Verstraelen, F. Heidar-Zadeh, and P. W. Ayers, "GBasis: A Python library for evaluating functions, functionals, and integrals expressed with Gaussian basis functions," J. Chem. Phys. **161**, 042503 (2024).

[16] A. Tehrani, J. S. M. Anderson, D. Chakraborty, J. I. Rodriguez-Hernandez, D. C. Thompson, T. Verstraelen, P. W. Ayers, and F. Heidar-Zadeh, "An information-theoretic approach to basis-set fitting of electron densities and other non-negative functions," J. Comput. Chem. **44**, 1998–2015 (2023).

[17] A. Tehrani, M. Richer, and F. Heidar-Zadeh, "CuGBasis: High-performance CUDA/Python library for efficient computation of quantum chemistry density-based descriptors for larger systems," J. Chem. Phys. **161**(7), 072501 (2024).

[18] T. D. Kim, M. Richer, G. Sánchez-Díaz, R. A. Miranda-Quintana, T. Verstraelen, F. Heidar-Zadeh, and P. W. Ayers, "Fanpy: A python library for prototyping multideterminant methods in *ab initio* quantum chemistry," J. Comput. Chem. **44**, 697–709 (2022).

[19] M. Richer, G. Sánchez-Díaz, M. Martínez-González, V. Chuiko, T. D. Kim, A. Tehrani, S. Wang, P. B. Gaikwad, C. E. V. de Moura, C. Masschelein, R. Alain Miranda-Quintana, A. Gerolin, F. Heidar-Zadeh, and P. W. Ayers, "PyCI: A python-scriptable library for arbitrary determinant ci," (Unpublished).

[20] A. Tehrani, X. D. Yang, M. Martínez-González, L. Pujal, R. Hernández-Esparza, M. Chan, E. Vöhringer-Martinez, T. Verstraelen, P. W. Ayers, and F. Heidar-Zadeh, "Grid: A python library for molecular integration, interpolation, differentiation, and more," J. Chem. Phys. **160**, 172503 (2024).

[21] F. Meng, M. Richer, A. Tehrani, J. La, T. D. Kim, P. W. Ayers, and F. Heidar-Zadeh, "Procrustes: A python library to find transformations that maximize the similarity between matrices," Comput. Phys. Commun. **276**, 108334 (2022).

[22] L. Pujal, A. Tehrani, and F. Heidar-Zadeh, "Chemtools: Gain chemical insight form quantum chemistry calculations," in *Conceptual Density Functional Theory: Towards a New Chemical Reactivity Theory*, 1st ed., edited by S. Liu (Wiley, 2022).

[23] F. Heidar-Zadeh, M. Richer, S. Fias, R. A. Miranda-Quintana, M. Chan, M. Franco-Perez, C. E. Gonzalez-Espinoza, T. D. Kim, C. Lanssens, A. H. G. Patel, X. D. Yang, E. Vohringer-Martinez, C. Cardenas, T. Verstraelen, and P. W. Ayers, "An explicit approach to conceptual density functional theory descriptors of arbitrary order," Chem. Phys. Lett. **660**, 307–312 (2016).

[24] Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma *et al.*, "PySCF: The python-based simulations of chemistry framework," WIREs Comput. Mol. Sci. **8**, e1340 (2018).

[25] Q. Sun, X. Zhang, S. Banerjee, P. Bao, M. Barbry, N. S. Blunt, N. A. Bogdanov, G. H. Booth, J. Chen, Z.-H. Cui, J. J. Eriksen, Y. Gao, S. Guo, J. Hermann, M. R. Hermes, K. Koh, P. Koval, S. Lehtola, Z. Li, J. Liu, N. Mardirossian, J. D. McClain, M. Motta, B. Mussard, H. Q. Pham, A. Pulkin, W. Purwanto, P. J. Robinson, E. Ronca, E. R. Sayfutyarova, M. Scheurer, H. F. Schurkus, J. E. T. Smith, C. Sun, S.-N. Sun, S. Upadhyay, L. K. Wagner, X. Wang, A. White, J. D. Whitfield, M. J. Williamson, S. Wouters, J. Yang, J. M. Yu, T. Zhu, T. C. Berkelbach, S. Sharma, A. Y. Sokolov, and G. K. L. Chan, "Recent developments in the PySCF program package," J. Chem. Phys. **153**, 024109 (2020).

[26] J. S. M. Anderson, F. Heidar-Zadeh, and P. W. Ayers, "Breaking the curse of dimension for the electronic Schrodinger equation with functional analysis," Comput. Theor. Chem. **1142**, 66–77 (2018).

[27] T. D. Kim, R. A. Miranda-Quintana, M. Richer, and P. W. Ayers, "Flexible ansatz for N-body configuration interaction," Comput. Theor. Chem. **1202**, 113187 (2021).

[28] P. A. Johnson, P. A. Limacher, T. D. Kim, M. Richer, R. A. Miranda-Quintana, F. Heidar-Zadeh, P. W. Ayers, P. Bultinck, S. De Baerdemacker, and D. Van Neck, "Strategies for extending geminal-based wavefunctions: Open shells and beyond," Comput. Theor. Chem. **1116**, 207–219 (2017).

[29] P. A. Johnson, P. W. Ayers, P. A. Limacher, S. D. Baerdemacker, D. V. Neck, and P. Bultinck, "A size-consistent approach to strongly correlated systems using a generalized antisymmetrized product of nonorthogonal geminals," Comput. Theor. Chem. **1003**, 101–113 (2013).

[30] P. A. Limacher, P. W. Ayers, P. A. Johnson, S. De Baerdemacker, D. Van Neck, and P. Bultinck, "A new mean-field method suitable for strongly correlated electrons: Computationally facile antisymmetric products of nonorthogonal geminals," J. Chem. Theory Comput. **9**, 1394–1401 (2013).

[31] P. B. Gaikwad, T. D. Kim, M. Richer, R. A. Lokhande, G. Sánchez-Díaz, P. A. Limacher, P. W. Ayers, and R. A. Miranda-Quintana, "Coupled cluster-inspired geminal wavefunctions," J. Chem. Phys. **160**, 144108 (2024).

[32] K. Jankowski and J. Paldus, "Applicability of coupled-pair theories to quasidegenerate electronic states: A model study," Int. J. Quantum Chem. **18**, 1243–1269 (1980).

[33] X. Z. Li and J. Paldus, "Comparison of the open-shell state-universal and state-selective coupled-cluster theories: H_4 and H_8 models," J. Chem. Phys. **103**, 1024–1034 (1995).

[34] J. Paldus, P. E. S. Wormer, and M. Benard, "Coupled-pair theories and Davidson-type corrections for quasidegenerate states: The H_4 model revisited," Collect. Czech. Chem. Commun. **53**, 1919–1942 (1988).

[35] A. Eugene DePrince III, "Variational determination of the two-electron reduced density matrix: A tutorial review," WIREs Comput. Mol. Sci. **14**, e1702 (2024).

[36] I. Magoulas, J. Shen, and P. Piecuch, "Addressing strong correlation by approximate coupled-pair methods with active-space and full treatments of three-body clusters," Mol. Phys. **120**, e2057365 (2022).

[37] N. H. Stair and F. A. Evangelista, "Exploring Hilbert space on a budget: Novel benchmark set and performance metric for testing electronic structure methods in the regime of strong correlation," J. Chem. Phys. **153**, 104108 (2020).

[38] J. Hachmann, W. Cardoen, and G. K. L. Chan, "Multireference correlation in long molecules with the quadratic scaling density matrix renormalization group," J. Chem. Phys. **125**, 144101 (2006).

[39] I. Mitxelena and M. Piris, "Benchmarking GNOF against FCI in challenging systems in one, two, and three dimensions," J. Chem. Phys. **156**, 214102 (2022).

[40] S. Wouters, P. A. Limacher, D. Van Neck, and P. W. Ayers, "Longitudinal static optical properties of hydrogen chains: Finite field extrapolations of matrix product state calculations," J. Chem. Phys. **136**, 134110 (2012).

[41] Y. Rath and G. H. Booth, "Framework for efficient *ab initio* electronic structure with Gaussian process states," Phys. Rev. B **107**, 205119 (2023).

[42] R. A. Miranda-Quintana, T. D. Kim, R. A. Lokhande, M. Richer, G. Sánchez-Díaz, P. B. Gaikwad, and P. W. Ayers, "Flexible ansatz for *N*-body perturbation theory," J. Phys. Chem. A **128**, 3458 (2024).

[43] C.-É. Fecteau, S. Cloutier, J.-D. Moisset, J. . Boulay, P. Bultinck, A. Faribault, and P. A. Johnson, "Near-exact treatment of seniority-zero ground and excited states with a Richardson–Gaudin mean-field," J. Chem. Phys. **156**, 194103 (2022).

[44] P. A. Johnson, C.-É. Fecteau, F. Berthiaume, S. Cloutier, L. Carrier, M. Gratton, P. Bultinck, S. De Baerdemacker, D. Van Neck, P. Limacher, and P. W. Ayers, "Richardson–Gaudin mean-field for strong correlation in quantum chemistry," J. Chem. Phys. **153**, 104110 (2020).

[45] K. Boguslawski, P. Tecmer, P. W. Ayers, P. Bultinck, S. De Baerdemacker, and D. Van Neck, "Efficient description of strongly correlated electrons with mean-field cost," Phys. Rev. B **89**, 201106 (2014).

[46] L. Bytautas, T. M. Henderson, C. A. Jiménez-Hoyos, J. K. Ellis, and G. E. Scuseria, "Seniority and orbital symmetry as tools for establishing a full configuration interaction hierarchy," J. Chem. Phys. **135**, 044119 (2011).

[47] J. K. Ellis, C. A. Jiménez-Hoyos, T. M. Henderson, T. Tsuchimochi, and G. E. Scuseria, "Constrained-pairing mean-field theory. V. Triplet pairing formalism," J. Chem. Phys. **135**, 034112 (2011).

[48] P. Andrew Johnson, "Richardson–Gaudin states," arXiv:2312.08804 [cond-mat, physics:physics] (2023).

[49] P. A. Johnson and A. E. DePrince III, "Single reference treatment of strongly correlated H_4 and H_{10} isomers with Richardson–Gaudin states," J. Chem. Theory Comput. **19**, 8129–8146 (2023).

[50] M. Motta, D. M. Ceperley, G. K.-L. Chan, J. A. Gomez, E. Gull, S. Guo, C. A. Jiménez-Hoyos, T. N. Lan, J. Li, F. Ma, A. J. Millis, N. V. Prokof'ev, U. Ray, G. E. Scuseria, S. Sorella, E. M. Stoudenmire, Q. Sun, I. S. Tupitsyn, S. R. White, D. Zgid, and S. Zhang, "Towards the solution of the many-electron problem in real materials: Equation of state of the hydrogen chain with state-of-the-art many-body methods," Phys. Rev. X **7**, 031059 (2017).

[51] T. Tsuchimochi and G. E. Scuseria, "Strong correlations via constrained-pairing mean-field theory," J. Chem. Phys. **131**, 121102 (2009).

[52] A. Glielmo, Y. Rath, G. . Csányi, A. De Vita, and G. H. Booth, "Gaussian process states: A data-driven representation of quantum many-body physics," Phys. Rev. X **10**, 041026 (2020).

[53] M. Rano and D. Ghosh, "Efficient machine learning configuration interaction for bond breaking problems," J. Phys. Chem. A **127**, 3705–3713 (2023).

[54] K. T. Schütt, P. Jan Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. Müller, "SchNet: A continuous-filter convolutional neural network for modeling quantum interactions," arXiv:1706.08566v5 (2017).

[55] 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).

[56] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and E. D. George, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, Proceedings of Machine Learning Research, edited by D. Precup and Y. W. Teh (ACM, 2017), Vol. 70, pp. 1263–1272.

[57] V. Chuiko and P. W. Ayers, "A size-consistent wave-function ansatz built from statistical analysis of orbital occupations," arXiv:2304.10484 [quant-ph] (2023).

[58] 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 *et al.*, "Array programming with NumPy," Nature **585**, 357–362 (2020).

[59] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in python," Nat. Methods **17**, 261–272 (2020).

[60] T. Helgaker, P. Jørgensen, and J. Olsen, *Molecular Electronic Structure Theory* (John Wiley & Sons, LTD, Chichester, 2000).

[61] P. Jordan and E. Wigner, "Über das Paulische Äquivalenzverbot," Z. Phys. **47**, 631–651 (1928).
[62] E. Lieb, T. Schultz, and D. Mattis, "Two soluble models of an antiferromagnetic chain," Ann. Phys. **16**, 407–466 (1961).
[63] M. A. Nielsen, "The fermionic canonical commutation relations and the Jordan–Wigner transform," (published online) (2005); available at https://futureofmatter.com/assets/fermions_and_jordan_wigner.pdf.
[64] J. T. Seeley, M. J. Richard, and P. J. Love, "The Bravyi–Kitaev transformation for quantum computation of electronic structure," J. Chem. Phys. **137**, 224109 (2012).
[65] A. Tranter, S. Sofia, J. Seeley, M. Kaicher, J. McClean, R. Babbush, P. V. Coveney, F. Mintert, F. Wilhelm, and P. J. Love, "The Bravyi–Kitaev transformation: Properties and applications," Int. J. Quantum Chem. **115**, 1431–1441 (2015).
[66] S. B. Bravyi and A. Y. Kitaev, "Fermionic quantum computation," Ann. Phys. **298**, 210–226 (2002).
[67] P. A. Johnson, P. W. Ayers, S. De Baerdemacker, P. A. Limacher, and D. Van Neck, "Bivariational principle for an antisymmetrized product of nonorthogonal geminals appropriate for strong electron correlation," Comput. Theor. Chem. **1212**, 113718 (2022).
[68] M. Gaudin, "Diagonalisation d'une classe d'hamiltoniens de spin," J. Phys. **37**, 1087–1098 (1976).
[69] M. C. Cambiaggio, A. M. F. Rivas, and M. Saraceno, "Integrability of the pairing Hamiltonian," Nucl. Phys. A **624**, 157–167 (1997).
[70] K. Boguslawski and P. W. Ayers, "Linearized coupled cluster correction on the antisymmetric product of 1-reference orbital geminals," J. Chem. Theory Comput. **11**, 5252–5261 (2015).
[71] K. Boguslawski, P. Tecmer, P. A. Limacher, P. A. Johnson, P. W. Ayers, P. Bultinck, S. De Baerdemacker, and D. Van Neck, "Projected seniority-two orbital optimization of the antisymmetric product of one-reference orbital geminal," J. Chem. Phys. **140**, 214114 (2014).
[72] A. Nowak, P. Tecmer, and K. Boguslawski, "Assessing the accuracy of simplified coupled cluster methods for electronic excited states in f0 actinide compounds," Phys. Chem. Chem. Phys. **21**, 19039–19053 (2019).
[73] P. Tecmer and K. Boguslawski, "Geminal-based electronic structure methods in quantum chemistry. Toward a geminal model chemistry," Phys. Chem. Chem. Phys. **24**, 23026 (2022).
[74] A. Faribault, C. Dimo, J.-D. Moisset, and P. A. Johnson, "Reduced density matrices/static correlation functions of Richardson–Gaudin states without rapidities," J. Chem. Phys. **157**, 214104 (2022).
[75] C.-É. Fecteau, H. Fortin, S. Cloutier, and P. A. Johnson, "Reduced density matrices of Richardson–Gaudin states in the Gaudin algebra basis," J. Chem. Phys. **153**, 164117 (2020).
[76] P. A. Johnson, H. Fortin, S. Cloutier, and C.-É. Fecteau, "Transition density matrices of Richardson–Gaudin states," J. Chem. Phys. **154**, 124125 (2021).
[77] J.-D. Moisset, C.-É. Fecteau, and P. A. Johnson, "Density matrices of seniority-zero geminal wavefunctions," J. Chem. Phys. **156**, 214110 (2022).
[78] T. M. Henderson, I. W. Bulik, T. Stein, and G. E. Scuseria, "Seniority-based coupled cluster theory," J. Chem. Phys. **141**, 244104 (2014).
[79] T. Stein, T. M. Henderson, and G. E. Scuseria, "Seniority zero pair coupled cluster doubles theory," J. Chem. Phys. **140**, 214113 (2014).
[80] R. J. Bartlett, "Coupled-cluster theory and its equation-of-motion extensions," WIREs Comput. Mol. Sci. **2**, 126–138 (2012).
[81] R. Chakraborty, K. Boguslawski, and P. Tecmer, "Static embedding with pair coupled cluster doubles based methods," Phys. Chem. Chem. Phys. **25**, 25377–25388 (2023).
[82] R. Dutta, F. Gao, A. Khamoshi, T. M. Henderson, and G. E. Scuseria, "Correlated pair ansatz with a binary tree structure," J. Chem. Phys. **160**, 084113 (2024).
[83] R. Dutta, T. M. Henderson, and G. E. Scuseria, "Geminal replacement models based on AGP," J. Chem. Theory Comput. **16**, 6358–6367 (2020).
[84] Z. Liu, F. Gao, G. P. Chen, T. M. Henderson, J. Dukelsky, and G. E. Scuseria, ""Exploring spin antisymmetrized geminal power Ansätze for strongly correlated spin systems," Phys. Rev. B **108**, 085136 (2023).
[85] M. Ravi, A. Perera, Y. C. Park, and R. J. Bartlett, "Excited states with pair coupled cluster doubles tailored coupled cluster theory," J. Chem. Phys. **159**, 094101 (2023).
[86] B. Senjean, S. Yalouz, N. Nakatani, and E. Fromager, "Reduced density matrix functional theory from an ab initio seniority-zero wave function: Exact and approximate formulations along adiabatic connection paths," Phys. Rev. A **106**, 032203 (2022).
[87] P. A. Limacher, "A new wavefunction hierarchy for interacting geminals," J. Chem. Phys. **145**, 194102 (2016).
[88] C. Lanssens, P. W. Ayers, D. Van Neck, S. De Baerdemacker, K. Gunst, and P. Bultinck, "Method for making 2-electron response reduced density matrices approximately N-representable," J. Chem. Phys. **148**, 084104 (2018).
[89] P. A. Limacher, P. W. Ayers, P. A. Johnson, S. De Baerdemacker, D. V. Neck, and P. Bultinck, "Simple and inexpensive perturbative correction schemes for antisymmetric products of nonorthogonal geminals," Phys. Chem. Chem. Phys. **16**, 5061–5065 (2014).
[90] W. Poelmans, M. Van Raemdonck, B. Verstichel, S. De Baerdemacker, A. Torre, L. Lain, G. E. Massaccesi, D. R. Alcoba, P. Bultinck, and D. Van Neck, "Variational optimization of the second-order density matrix corresponding to a seniority-zero configuration interaction wave function," J. Chem. Theory Comput. **11**, 4064–4076 (2015).
[91] M. Van Raemdonck, D. R. Alcoba, W. Poelmans, S. De Baerdemacker, A. Torre, L. Lain, G. E. Massaccesi, D. Van Neck, and P. Bultinck, "Polynomial scaling approximations and dynamic correlation corrections to doubly occupied configuration interaction wave functions," J. Chem. Phys. **143**, 104106 (2015).
[92] N. Vu and A. E. DePrince III, "Size-extensive seniority-zero energy functionals derived from configuration interaction with double excitations," J. Chem. Phys. **152**, 244103 (2020).
[93] K. Boguslawski, F. Brzek, R. Chakraborty, K. Cieślak, S. Jahani, A. Leszczyk, A. Nowak, E. Sujkowski, J. Świerczyński, S. Ahmadkhani, D. Kedziera, M. H. Kriebel, P. S. Żuchowski, and P. Tecmer, "PyBEST: Improved functionality and enhanced performance," Comput. Phys. Commun. **297**, 109049 (2024).
[94] K. Boguslawski, A. Leszczyk, A. Nowak, F. Brzęk, P. S. Żuchowski, D. Kędziera, and P. Tecmer, "Pythonic Black-box Electronic Structure Tool (PyBEST). An open-source python platform for electronic structure calculations at the interface between chemistry and physics," Comput. Phys. Commun. **264**, 107933 (2021).
[95] T. M. Henderson, G. P. Chen, and G. E. Scuseria, "Strong–weak duality via Jordan–Wigner transformation: Using fermionic methods for strongly correlated $su(2)$ spin systems," J. Chem. Phys. **157**, 194114 (2022).
[96] A. Rauk, *Orbital Interaction Theory of Organic Chemistry* (Wiley, 2000).
[97] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," arXiv:2005.14165 [cs.CL] (2020).
[98] T. Preston-Werner and P. Gedam, Tom's obvious, minimal language (2021) https://toml.io/en/v1.0.0
[99] R. N. Tazhigulov, S.-N. Sun, R. Haghshenas, H. Zhai, A. T. K. Tan, N. C. Rubin, R. Babbush, A. J. Minnich, and G. K. L. Chan, "Simulating models of challenging correlated molecules and materials on the sycamore quantum processor," PRX Quantum **3**, 040318 (2022).
[100] D. Weininger, "Smiles, a chemical language and information system. 1. Introduction to methodology and encoding rules," J. Chem. Inf. Comput. Sci. **28**, 31–36 (1988).
[101] A. Dalby, J. G. Nourse, W. D. Hounshell, A. K. I. Gushurst, D. L. Grier, B. A. Leland, and J. Laufer, "Description of several chemical structure file formats used by computer programs developed at molecular design limited," J. Chem. Inf. Comput. Sci. **32**, 244–255 (1992).
[102] G. Landrum, P. Tosco, B. Kelley, R. Rodriguez, D. Cosgrove, R. V. Sriniker, N. S. Gedeck, G. Jones, E. Kawashima, D. Nealschneider, A. Dalke, B. Cole, M. Swain, S. Turk, A. Savelev, A. Vaucher, M. Wójcikowski, I. Take, V. F. Scalfani, R. Walker, K. Ujihara, D. Probst, G. Godin, A. Pahl, J. Lehtivarjo, and F.

Berenger (2024). "jasondbiggs, and strets123, "rdkit/rdkit: 2024_03_1 (q1 2024) release," Zenodo. https://doi.org/10.5281/zenodo.591637
[103] L. Lemmens, X. De Vriendt, D. Van Hende, T. Huysentruyt, P. Bultinck, and G. Acke, "GQCP: The Ghent quantum chemistry package," J. Chem. Phys. **155**, 084802 (2021).
[104] M. Abram, M. Zegrodnik, and J. Spałek, "Antiferromagnetism, charge density wave, and *d*-wave superconductivity in the extended t–J–U model: Role of intersite coulomb interaction and a critical overview of renormalized mean field theory," J. Phys.: Condens. Matter **29**, 365602 (2017).
[105] C. Tarafdar, N. K. Ghosh, and S. Nath, "Role of inter-site Coulomb interaction on the thermodynamic and ground state properties within the t–J–U–V model," Physica C **615**, 1354393 (2023).
[106] K. A. Chao, J. Spałek, and A. M. Oleś, "Canonical perturbation expansion of the Hubbard model," Phys. Rev. B **18**, 3453–3464 (1978).
[107] F. C. Zhang and T. M. Rice, "Effective Hamiltonian for the superconducting Cu oxides," Phys. Rev. B **37**, 3759 (1988).
[108] M. Ogata and H. Fukuyama, "The t–J model for the oxide high-T_c superconductors," Rep. Prog. Phys. **71**, 036501 (2008).