
The Three Qiskiteers H_2 ground state finder

Brett Henderson, Igor Benek-Lins and Melvin Mathews

Feb 04, 2021

Contents:

I	API Reference	I
1.1	<code>solver</code>	1
1.2	<code>mapping</code>	2
1.3	<code>pauli_string</code>	3
1.4	<code>hamiltonian</code>	7
1.5	<code>evaluator</code>	10
1.6	<code>sim_noisy</code>	12
1.7	<code>sim_noisy_layout_opt</code>	13
1.8	<code>sim_no_noise</code>	13
1.9	<code>dissociation_curve</code>	14
1.10	<code>sim_noisy_measfilt_layout_opt</code>	15
1.11	<code>conf</code>	15
2	Indices and tables	17
	Python Module Index	19
	Index	21

This page contains auto-generated API reference documentation¹.

1.1 solver

solver.py - Solvers for LinearCombinaisonPauliString

Copyright © 2021 Brett Henderson <brettrhenderson25@gmail.com>, Igor Benek-Lins <physics@ibeneklins.com>, Melvin Mathews <mel.mattoo7@gmail.com>,

Copyright © 2020-2021 Maxime Dion <maxime.dion@usherbrooke.ca>

This file is part of “The Three Qiskiteers H₂ ground state finder” (T₃QH₂). For the licence, see the LICENCE file.

1.1.1 Module Contents

Classes

LCPSSolver

ExactSolver

VQESolver

class solver.**LCPSSolver**

Bases: object

class solver.**ExactSolver**

Bases: *solver.LCPSSolver*

eig(*self*, *lcps*)

Convert LCPS into a matrix and return sorted eigenvalues and eigenvectors.

lcps [LinearCombinaisonPauliString] The LCPS to be solved.

np.array, np.array Eigenvalues and eigenvectors sorted with respect to the eigenvalues.

¹ Created with sphinx-autoapi²

² <https://github.com/readthedocs/sphinx-autoapi>

lowest_eig_value(*self*, *lcps*)

Return lowest eigenvalue and the associated eigenvector.

lcps [LinearCombinaisonPauliString] The LCPS to be solved.

float, np.array The lowest eigenvalue and the associated eigenvector.

class solver.VQESolver(*evaluator*, *minimizer*, *start_params*, *name*='vqe_solver')

Bases: [solver.LCPSSolver](#)

lowest_eig_value(*self*, *lcps*)

Return lowest expectation value and associated parameters the minimization could find.

lcps [LinearCombinaisonPauliString] The LCPS to be solved.

float, np.array The lowest eigenvalue and the associated parameters.

1.2 mapping

mapping.py - Map a Hamiltonian to a LinearCombinaisonPauliString

Copyright © 2021 Brett Henderson <brettrhenderson25@gmail.com>, Igor Benek-Lins <physics@ibeneklins.com>, Melvin Mathews <mel.mattoo7@gmail.com>,

Copyright © 2020-2021 Maxime Dion <maxime.dion@usherbrooke.ca>

This file is part of “The Three Qiskiteers H₂ ground state finder” (T₃QH₂). For the licence, see the LICENCE file.

1.2.1 Module Contents

Classes

[Mapping](#)

[JordanWigner](#)

[Parity](#)

class mapping.Mapping

Bases: `object`

fermionic_hamiltonian_to_linear_combinaison_pauli_string(*self*, *fermionic_hamiltonian*)

Do the mapping of a *FermionicHamiltonian*. First generates the LCPS representation of the creation/annihilation operators for the specific mapping. Uses the *to_linear_combinaison_pauli_string* of the *FermionicHamiltonian* to generate the complete LCPS.

fermionic_hamiltonian [FermionicHamiltonian] A *FermionicHamiltonian* that provided *to_linear_combinaison_pauli_string* method.

LinearCombinaisonPauliString The LCPS representing the *FermionicHamiltonian*.

class mapping.JordanWigner

Bases: [mapping.Mapping](#)

fermionic_operator_linear_combinaison_pauli_string(*self*, *n_qubits*)

Build the LCPS representations for the creation/annihilation operator for each qubit following Jordan-Wigner mapping.

n_qubits [int] The number of orbitals to be mapped to the same number of qubits.

list<LinearCombinaisonPauliString>, list<LinearCombinaisonPauliString> Lists of the creation/annihilation operators for each orbital in the form of *LinearCombinaisonPauliString*.

class mapping.Parity

Bases: *mapping.Mapping*

fermionic_operator_linear_combinaison_pauli_string(*self*, *n_qubits*)

Build the LCPS representations for the creation/annihilation operator for each qubit following the parity mapping.

n_qubits [int] The number of orbitals to be mapped to the same number of qubits.

list<LinearCombinaisonPauliString>, list<LinearCombinaisonPauliString> Lists of the creation/annihilation operators for each orbital in the form of *LinearCombinaisonPauliString*.

1.3 pauli_string

pauli_string.py - Define PauliString and LinearCombinaisonPauliString

Copyright © 2021 Brett Henderson <brettrhenderson25@gmail.com>, Igor Benek-Lins <physics@ibeneklins.com>, Melvin Mathews <mel.mattoo7@gmail.com>,

Copyright © 2020-2021 Maxime Dion <maxime.dion@usherbrooke.ca>

This file is part of “The Three Qiskiteers H₂ ground state finder” (T₃QH₂). For the licence, see the LICENCE file.

1.3.1 Module Contents

Classes

PauliString

LinearCombinaisonPauliString

class pauli_string.PauliString(*z_bits*, *x_bits*)

Bases: object

__str__(*self*)

String representation of the *PauliString*.

str String of I, Z, X and Y.

__len__(*self*)

Number of Pauli in the *PauliString*. It is also the number of qubits.

int Length of the *PauliString*, also number of qubits.

__mul__(*self*, *other*)

Allow the use of * with other *PauliString* or with a numeric coefficient.

other [PauliString/float] Will compute the product of Pauli strings if *PauliString* or computer a linear combination of Pauli strings if *float*.

PauliString/LinearCombinaisonPauliString *PauliString* when other is a *PauliString* or *LinearCombinaisonPauliString* when other is numeric

__rmul__(*self*, *other*)

Same as **__mul__**. Allow the use of `*` with a preceding numeric coefficient. Example: `1/2 * PauliString`.

other [*PauliString*/float] Will compute the product of Pauli strings if *PauliString* or computer a linear combination of Pauli strings if *float*.

PauliString/LinearCombinaisonPauliString *PauliString* when other is a *PauliString* or *LinearCombinaisonPauliString* when other is numeric

classmethod from_zx_bits(*cls*, *zx_bits*)

Construct a *PauliString* from a single *array<bool>* of length $2n$.

zx_bits [*np.array<bool>*] An array of booleans. First n bits specify the `:math:'Z's`. Second half specify the `:math:'X's`.

PauliString The Pauli string specified by *zx_bits*.

classmethod from_str(*cls*, *pauli_str*)

Construct a *PauliString* from a string (as returned by **__str__**).

pauli_str [*str*] String of length n made of *I*, *X*, *Y* and *Z*.

PauliString The Pauli string specified by *pauli_str*.

to_zx_bits(*self*)

Return the *zx_bits* representation of the *PauliString*. Useful to compare `'PauliString's` together.

np.array<bool> *zx_bits* representation of the *PauliString* of length $2n$.

to_xz_bits(*self*)

Return the *xz_bits* representation of the *PauliString*. Useful to check commutativity.

np.array<bool> *xz_bits* representation of the *PauliString* of length $2n$.

mul_pauli_string(*self*, *other*)

Product with an *other* *Pauli* string.

other [*PauliString*] An other *PauliString*.

ValueError: If the other *PauliString* is not of the same length.

PauliString [*complex*] The resulting *PauliString* and the product phase.

mul_coef(*self*, *coef*)

Build a LCPS from a *PauliString* (*self*) and a number (*coef*).

coef [*int/float/complex*] A numeric coefficient.

LinearCombinaisonPauliString A LCPS with only one *PauliString* and coefficient.

ids(*self*)

Position of identity in the *PauliString*.

np.array<bool> True where both *z_bits* and *x_bits* are *False*.

copy(*self*)

Build a copy of the *PauliString*.

PauliString A copy.

to_matrix(*self*)
Build the matrix representation of the *PauliString* using the Kronecker product.

np.array<complex> A 2^n side square matrix.

class pauli_string.LinearCombinaisonPauliString(*coefs, pauli_strings*)
Bases: object

__str__(*self*)
String representation of the *LinearCombinaisonPauliString*.

str Descriptive string.

__getitem__(*self, key*)
Return a subset of the *LinearCombinaisonPauliString* array-like.

key [int or slice] Elements to be returned.

LinearCombinaisonPauliString LCPS with the element specified in key.

__len__(*self*)
Number of *PauliStrings* in the LCPS.

int Number of *PauliStrings*/coefficients.

__add__(*self, other*)
Allow the use of + to add two LCPS together.

other [LinearCombinaisonPauliString] An other LCPS.

LinearCombinaisonPauliString New LCPS of length $\text{len}(\text{self}) + \text{len}(\text{other})$.

__mul__(*self, other*)
Allow the use of * with other LCPS or numeric values.

other [LinearCombinaisonPauliString] An other LCPS

LinearCombinaisonPauliString/LinearCombinaisonPauliString New LCPS of length $\text{len}(\text{self}) * \text{len}(\text{other})$ or a new LCPS of same length with modified coefficients.

__rmul__(*self, other*)
Same as **__mul__**. Allow the use of * with a preceding numeric coefficient. Example: $1/2 * \text{PauliString}$.

other [LinearCombinaisonPauliString] An other LCPS.

LinearCombinaisonPauliString/LinearCombinaisonPauliString New LCPS of length $\text{len}(\text{self}) * \text{len}(\text{other})$ or a new LCPS of same length with modified coefficients.

add_pauli_string_linear_combinaison(*self, other*)
Adding with an other LCPS. Merging the coefficients and *PauliStrings* arrays.

other [LinearCombinaisonPauliString] An other LCPS.

ValueError If *other* is not an LCPS.

ValueError If the other LCPS has not the same number of qubits.

LinearCombinaisonPauliString New LCPS of length $\text{len}(\text{self}) + \text{len}(\text{other})$.

mul_linear_combinaison_pauli_string(*self*, *other*)

Multiply with an other LCPS.

other [LinearCombinaisonPauliString] An other LCPS.

ValueError If *other* is not an LCPS.

ValueError If the other LCPS has not the same number of qubits.

LinearCombinaisonPauliString New LCPS of length $\text{len}(\text{self}) * \text{len}(\text{other})$.

mul_coef(*self*, *other*)

Multiply the LCPS by a numeric coefficient or an array of the same length.

other [float/complex/np.array] One numeric factor or one factor per *PauliString*.

ValueError If *other* is *np.array*, it should be of the same length as the LCPS.

LinearCombinaisonPauliString New LCPS properly multiplied by the coefficients.

to_zx_bits(*self*)

Build an array that contains all the *zx_bits* for each *PauliString*.

np.array<bool> A two-dimensional array of booleans where each line is the *zx_bits* of a *PauliString*.

to_xz_bits(*self*)

Build an array that contains all the *xz_bits* for each *PauliString*.

np.array<bool> A two-dimensional array of booleans where each line is the *xz_bits* of a *PauliString*.

ids(*self*)

Build an array that identifies the position of all the *I* for each *PauliString*.

np.array<bool> A two-dimensional array of booleans where each line is the *xz_bits* of a *PauliString*.

combine(*self*)

Finds unique *PauliStrings* in the LCPS and combines the coefficients of identical *PauliStrings*. Reduces the length of the LCPS.

LinearCombinaisonPauliString LCPS with combined coefficients.

apply_threshold(*self*, *threshold*=1e-06)

Remove *PauliStrings* with coefficients smaller than threshold.

threshold [float, optional, default=1e-6] *PauliStrings* with coefficients smaller than *threshold* will be removed.

LinearCombinaisonPauliString LCPS without coefficients smaller than threshold.

divide_in_bitwise_commuting_cliques(*self*)

Find bitwise commuting cliques in the LCPS.

list<LinearCombinaisonPauliString> List of LCPS where all elements of one LCPS bitwise commute with each other.

sort(*self*)

Sort the *PauliStrings* by order of the *zx_bits*.

LinearCombinaisonPauliString Sorted.

to_matrix(*self*)

Build the total matrix representation of the LCPS.

`np.array<complex>` A $2 * n$ side square matrix.

1.4 hamiltonian

hamiltonian.py - Define the Hamiltonian

Copyright © 2021 Brett Henderson <brettrhenderson25@gmail.com>, Igor Benek-Lins <physics@ibeneklins.com>, Melvin Mathews <mel.mattoo7@gmail.com>,

Copyright © 2020-2021 Maxime Dion <maxime.dion@usherbrooke.ca>

This file is part of “The Three Qiskiteers H₂ ground state finder” (T₃QH₂). For the licence, see the LICENCE file.

1.4.1 Module Contents

Classes

<i>FermionicHamiltonian</i>
<i>OneBodyFermionicHamiltonian</i>
<i>TwoBodyFermionicHamiltonian</i>
<i>MolecularFermionicHamiltonian</i>

class hamiltonian.FermionicHamiltonian

Bases: object

__str__(*self*)

String representation of *FermionicHamiltonian*.

str Description of *FermionicHamiltonian*.

number_of_orbitals(*self*)

Number of orbitals in the state basis.

int The number of orbitals in the state basis.

include_spin(*self*, *order*=*'group_spin'*)

Transforms a spinless *FermionicHamiltonian* to include spin. The transformation doubles the number of orbitals in the basis following the input order. Does nothing if the spin is already included (*with_spin* is *True*).

order [str, optional, {*group_spin*, *group_orbital*}] Controls the order of the basis state. With *order* as *group_orbital*, the integrals will alternate between spin up and down (*g_{up}*, *g_{down}*, ...). With *order* as *group_spin*, the integrals will gather same spin together (*g_{up}*, ..., *g_{down}*, ...).

ValueError If the *order* parameter is not one of *group_spin* or *group_orbital*.

FermionicHamiltonian Including the spin.

get_integrals(*self*, *cut_zeros*=*True*, *threshold*=*1e-09*)

Returns the integral tensor with an optional threshold for values close to 0.

cut_zeros [bool, optional, default=*True*] If *True*, all integral values smaller than *threshold* will be set to 0.

threshold [float, optional, default=*1e-9*.] Value of the threshold.

np.ndarray The integral tensor.

class hamiltonian.**OneBodyFermionicHamiltonian**(*integrals*, *with_spin=False*)

Bases: [hamiltonian.FermionicHamiltonian](#)

spin_tensor

change_basis(*self*, *transform*)

Transforms the integrals tensor ($n*n$) into a new basis.

transform [np.ndarray] Square tensor ($n*n$) defining the basis change.

OneBodyFermionicHamiltonian Transformed Hamiltonian.

to_linear_combinaison_pauli_string(*self*, *aps*, *ams*)

Generates a qubit operator representation (*LinearCombinaisonPauliString*) of the *OneBodyFermionicHamiltonian* given some creation/annihilation operators.

aps [list<LinearCombinaisonPauliString>] List of the creation operators for each orbital in the form of *LinearCombinaisonPauliString*.

ams [list<LinearCombinaisonPauliString>] List of the annihilation operators for each orbital in the form of *LinearCombinaisonPauliString*.

LinearCombinaisonPauliString Qubit operator representation of the *OneBodyFermionicHamiltonian*.

class hamiltonian.**TwoBodyFermionicHamiltonian**(*integrals*, *with_spin=False*)

Bases: [hamiltonian.FermionicHamiltonian](#)

spin_tensor

change_basis(*self*, *transform*)

Transforms the integrals tensor ($n*n*n*n$) into a new basis.

transform [np.ndarray] Square tensor ($n*n$) defining the basis change.

TwoBodyFermionicHamiltonian Transformed Hamiltonian.

to_linear_combinaison_pauli_string(*self*, *aps*, *ams*)

Generates a qubit operator representation (*LinearCombinaisonPauliString*) of the *TwoBodyFermionicHamiltonian* given some creation/annihilation operators.

aps [list<LinearCombinaisonPauliString>] List of the creation operators for each orbital in the form of *LinearCombinaisonPauliString*.

ams [list<LinearCombinaisonPauliString>] List of the annihilation operators for each orbital in the form of *LinearCombinaisonPauliString*.

LinearCombinaisonPauliString Qubit operator representation of the *TwoBodyFermionicHamiltonian*.

class hamiltonian.**MolecularFermionicHamiltonian**(*one_body*, *two_body*, *with_spin=False*)

Bases: [hamiltonian.FermionicHamiltonian](#)

classmethod from_integrals(*cls*, *h1*, *h2*)

Generates a *MolecularFermionicHamiltonian* describing a molecule from *h1* and *h2* integral tensors.

h1 [np.ndarray(n,n)] One body integral tensor

h2 [np.ndarray(n,n,n,n)] Two Body integral tensor

MolecularFermionicHamiltonian: The Hamiltonian describing the molecule including one *OneBody* and one *TwoBody* terms.

classmethod from_pyscf_mol(cls, mol)

Generates a *MolecularFermionicHamiltonian* describing a molecule from a pyscf molecule representation.

mol [pyscf.gto.mole.Mole] Molecule object used to compute different integrals.

MolecularFermionicHamiltonian The Hamiltonian describing the molecule including one *OneBody* and one *TwoBody* terms.

number_of_orbitals(self)

Number of orbitals in the state basis.

int The number of orbitals in the state basis.

change_basis(self, transform)

Transforms the integrals tensors for both sub Hamiltonian. See *FermionicHamiltonian.change_basis*.

transform [np.ndarray] Square tensor ($n \times n$) defining the basis change.

MolecularFermionicHamiltonian Transformed Hamiltonian.

include_spin(self)

Transforms a spinless *FermionicHamiltonian* to include spin for both sub Hamiltonians. See *FermionicHamiltonian.include_spin*.

order [str, optional, {group_spin, group_orbital}] Controls the order of the basis state. With order as *group_orbital*, the integrals will alternate between spin up and down (g_{up}, g_{down}, \dots). With order as *group_spin*, the integrals will gather same spin together ($g_{up}, \dots, g_{down}, \dots$).

ValueError If the order parameter is not one of *group_spin* or *group_orbital*.

FermionicHamiltonian Including the spin.

get_integrals(self, **vargs)

Return the integral tensors for both sub Hamiltonians with an optional threshold for values close to 0.

cut_zeros [bool, optional, default=True] If *True*, all integral values smaller than *threshold* will be set to 0.

threshold [float, optional, default=1e-9] Value of the threshold.

np.ndarray, np.ndarray The integral tensors.

to_linear_combinaison_pauli_string(self, aps, ams)

Generates a qubit operator representation (*LinearCombinaisonPauliString*) of the *MolecularFermionicHamiltonian* given some creation/annihilation operators.

aps [list<LinearCombinaisonPauliString>] List of the creation operators for each orbital in the form of *LinearCombinaisonPauliString*.

ams [list<LinearCombinaisonPauliString>] List of the annihilation operators for each orbital in the form of *LinearCombinaisonPauliString*.

LinearCombinaisonPauliString Qubit operator representation of the *MolecularFermionicHamiltonian*.

1.5 evaluator

evaluator.py - Evaluate LinearCombinaisonPauliString on state functions (quantum circuit)

Copyright © 2021 Brett Henderson <brettrhenderson25@gmail.com>, Igor Benek-Lins <physics@ibeneklins.com>, Melvin Mathews <mel.mattoo7@gmail.com>,

Copyright © 2020-2021 Maxime Dion <maxime.dion@usherbrooke.ca>

This file is part of “The Three Qiskiteers H₂ ground state finder” (T₃QH₂). For the licence, see the LICENCE file.

1.5.1 Module Contents

Classes

<i>Evaluator</i>	
<i>BasicEvaluator</i>	The <i>BasicEvaluator</i> should build 1 quantum circuit and 1 interpreter for
<i>BitwiseCommutingCliqueEvaluator</i>	The <i>BitwiseCommutingCliqueEvaluator</i> should build 1 quantum circuit and

class evaluator.**Evaluator**(*varform*, *backend*, *execute_opts*={}, *measure_filter*=None, *record*=None)

Bases: object

set_linear_combinaison_pauli_string(*self*, *lcps*)

Set the LCPS to be evaluated. Further LCPS can be later provided still using the same Evaluator. This sets the value of the attribute *n_qubits*. The measurement circuits and the interpreters are generated right away with *prepare_measurement_circuits_and_interpreters* (defined at the subclass level).

lcps [LinearCombinaisonPauliString] The LCPS to be evaluated.

eval(*self*, *params*)

Evaluate an estimate of the expectation value of the set LCPS.

params [list or np.array] Parameter values at which the expectation value should be evaluated. Will be fed to the *varform* parametrised QuantumCircuit.

float The value of the estimated expectation value of the LCPS.

prepare_eval_circuits(*self*, *params*)

Assign parameter values to the variational circuit (*varform*) to set the wave function. Combine *varform* circuit with each of the measurement circuits.

params [list or np.array] Parameters to be assigned to the *varform* QuantumCircuit.

list<QuantumCircuit> All the QuantumCircuit necessary to the evaluation of the LCPS.

counts2array(*self*, *counts*)

Transform a counts dictionary into an array.

counts [dict] The counts dict as return by *qiskit.result.Result.get_counts()*.

np.array<int> Counts vector sorted in the usual way: 0...00, 0...01, 0...10, ..., 1...11

interpret_count_arrays(*self*, *counts_arrays*)

Interprets all the *counts_arrays* resulting from the execution of all the *eval_circuits*. This computes the $\sum_i h_i < P_i >$.

counts_arrays [list<np.array>] *counts_arrays* resulting from the execution of all the *eval_circuits*.

float Sum of all the interpreted values of *counts_arrays*. Mathematically returns $\sum_i h_i < P_i >$.

static interpret_count_array(*interpreter*, *counts_array*)

Interprets the *counts_array* resulting from the execution of one *eval_circuit*. This computes the $h_i < P_i >$ either for one *PauliString* or a *Clique*.

interpreter [np.array] Array of the eigenvalues of the measurable *PauliStrings* associated with the circuit that returned the *counts_array*.

counts_array [np.array] *count_arrays* resulting from the execution of the *eval_circuit* associated with the *interpreter*.

float the interpreted values for the *PauliStrings* associated with the *eval_circuit* that gave *counts_arrays*. Mathematically returns $h_i < P_i >$.

static pauli_string_based_measurement(*pauli_string*)

Build a *QuantumCircuit* that measures the qubits in the basis given by a *PauliString*.

pauli_string : *PauliString*

qiskit.QuantumCircuit A quantum circuit starting with rotation of the qubit following the *PauliString* and finishing with the measurement of all the qubits.

static measurable_eigenvalues(*pauli_string*)

Build the eigenvalues vector (size = 2^{**n_qubits}) for the measurable version of a given *PauliString*.

pauli_string : *PauliString*

np.array<int> The eigenvalues.

class evaluator.BasicEvaluator(*varform*, *backend*, *execute_opts*={}, *measure_filter*=None, *record*=None)

Bases: *evaluator.Evaluator*

The *BasicEvaluator* should build 1 quantum circuit and 1 interpreter for each *PauliString*. The interpreter should be a one-dimensional array of size $2^{**number\ of\ qubits}$. It does not exploit the fact that commuting *PauliStrings* can be evaluated from a common circuit.

static prepare_measurement_circuits_and_interpreters(*lcps*)

For each *PauliString* in the LCPS, this method build a measurement *QuantumCircuit* and provide the associated interpreter. This interpreter allows to compute $h < P > = \sum_i T_i N_i / N_{tot}$ for each *PauliString*.

lcps [LinearCombinaisonPauliString] The LCPS to be evaluated, being set.

list<qiskit.QuantumCircuit>, list<np.array>

static pauli_string_circuit_and_interpreter(*coef*, *pauli_string*)

This method builds a measurement *QuantumCircuit* for a *PauliString* and provide the associated interpreter. The interpreter includes the associated coefficient for convenience.

coef [complex or float] The coefficient associated to the *pauli_string*.

pauli_string [*PauliString*] *PauliString* to be measured and interpreted.

qiskit.QuantumCircuit, np.array The *QuantumCircuit* to be used to measure in the basis given by the *PauliString* given with the interpreter to interpret to result of the eventual *eval_circuit*.

class evaluator.**BitwiseCommutingCliqueEvaluator**(*varform*, *backend*, *execute_opts*={}, *measure_filter*=None, *record*=None)

Bases: `evaluator.Evaluator`

The *BitwiseCommutingCliqueEvaluator* should build 1 quantum circuit and 1 interpreter for each clique of *PauliStrings*. The interpreter should be a two-dimensional array of size (*number of cliques*, *2**number of qubits*). It does exploit the fact that commuting *PauliStrings* can be evaluated from a common circuit.

static **prepare_measurement_circuits_and_interpreters**(*lcps*)

Divide the LCPS into bitwise commuting cliques. For each *PauliString* clique in the LCPS, this method builds a measurement *QuantumCircuit* and provides the associated interpreter. This interpreter allows to compute $\sum_i h_i \langle P_i \rangle = \sum_j T_j N_j / N_{tot}$ for each *PauliString* clique.

lcps [LinearCombinaisonPauliString] The LCPS to be evaluated, being set.

list<qiskit.QuantumCircuit>, **list<np.array>** The *QuantumCircuit* to be used to measure in the basis given by the *PauliString* clique given with the interpreter to interpret to result of the eventual *eval_circuit*.

static **bitwise_clique_circuit_and_interpreter**(*clique*)

This method builds a measurement *QuantumCircuit* for a *PauliString* clique and provides the associated interpreter. The interpreter includes the associated coefficients for convenience.

clique [LinearCombinaisonPauliString] A LCPS where all *PauliString* bitwise commute with another.

qiskit.QuantumCircuit, **np.array** The *QuantumCircuit* to be used to measure in the basis given by the *PauliString* clique given with the interpreter to interpret to result of the eventual *eval_circuit*.

1.6 sim_noisy

1.6.1 Module Contents

Functions

`get_energies`(N, shots)

`sim_noisy.provider`

`sim_noisy.bogota`

`sim_noisy.bogota_prop`

`sim_noisy.bogota_conf`

`sim_noisy.bogota_nm`

`sim_noisy.varform_4qubits_1param`

`sim_noisy.a`

`sim_noisy.out`

`sim_noisy.molecular_hamiltonian`

`sim_noisy.mapping`

`sim_noisy.lcps_h2`

`sim_noisy.qasm_simulator`


```
sim_noisy.minimizer
sim_noisy.get_energies(N, shots)
sim_noisy.N = 50
```

1.7 sim_noisy_layout_opt

1.7.1 Module Contents

Functions

```
get_energies(N, shots)
```

```
sim_noisy_layout_opt.provider
sim_noisy_layout_opt.bogota
sim_noisy_layout_opt.bogota_prop
sim_noisy_layout_opt.bogota_conf
sim_noisy_layout_opt.bogota_nm
sim_noisy_layout_opt.varform_4qubits_1param
sim_noisy_layout_opt.a
sim_noisy_layout_opt.out
sim_noisy_layout_opt.molecular_hamiltonian
sim_noisy_layout_opt.mapping
sim_noisy_layout_opt.lcps_h2
sim_noisy_layout_opt.qasm_simulator
sim_noisy_layout_opt.minimizer
sim_noisy_layout_opt.get_energies(N, shots)
sim_noisy_layout_opt.N = 50
```

1.8 sim_no_noise

1.8.1 Module Contents

Functions

```
get_energies(N, shots)
```

```
sim_no_noise.varform_4qubits_1param
sim_no_noise.a
sim_no_noise.out
```

```
sim_no_noise.molecular_hamiltonian
sim_no_noise.mapping
sim_no_noise.lcps_h2
sim_no_noise.qasm_simulator
sim_no_noise.minimizer
sim_no_noise.get_energies(N, shots)
sim_no_noise.N = 50
```

1.9 dissociation_curve

1.9.1 Module Contents

Functions

get_energies(*N*, *shots*, *distance*)

```
dissociation_curve.provider
dissociation_curve.bogota
dissociation_curve.bogota_prop
dissociation_curve.bogota_conf
dissociation_curve.bogota_nm
dissociation_curve.qasm_simulator
dissociation_curve.qr
dissociation_curve.qubit_list = [0, 1, 2, 3]
dissociation_curve.calibration_layout = [2, 3, 1, 4]
dissociation_curve.result
dissociation_curve.meas_fitter
dissociation_curve.meas_filter
dissociation_curve.varform_4qubits_1param
dissociation_curve.a
dissociation_curve.minimizer
dissociation_curve.get_energies(N, shots, distance)
dissociation_curve.shots = 1024
```

1.10 sim_noisy_measfilt_layout_opt

1.10.1 Module Contents

Functions

`get_energies(N, shots)`

```

sim_noisy_measfilt_layout_opt.provider
sim_noisy_measfilt_layout_opt.bogota
sim_noisy_measfilt_layout_opt.bogota_prop
sim_noisy_measfilt_layout_opt.bogota_conf
sim_noisy_measfilt_layout_opt.bogota_nm
sim_noisy_measfilt_layout_opt.qasm_simulator
sim_noisy_measfilt_layout_opt.qr
sim_noisy_measfilt_layout_opt.qubit_list = [0, 1, 2, 3]
sim_noisy_measfilt_layout_opt.calibration_layout = [2, 3, 1, 4]
sim_noisy_measfilt_layout_opt.result
sim_noisy_measfilt_layout_opt.meas_fitter
sim_noisy_measfilt_layout_opt.meas_filter
sim_noisy_measfilt_layout_opt.varform_4qubits_1param
sim_noisy_measfilt_layout_opt.a
sim_noisy_measfilt_layout_opt.out
sim_noisy_measfilt_layout_opt.molecular_hamiltonian
sim_noisy_measfilt_layout_opt.mapping
sim_noisy_measfilt_layout_opt.lcps_h2
sim_noisy_measfilt_layout_opt.minimizer
sim_noisy_measfilt_layout_opt.get_energies(N, shots)
sim_noisy_measfilt_layout_opt.N = 50

```

1.11 conf

1.11.1 Module Contents

```

conf.project = The Three Qiskiteers H2 ground state finder
conf.copyright = 2021, Brett Henderson, Igor Benek-Lins and Melvin Mathews
conf.author = Brett Henderson, Igor Benek-Lins and Melvin Mathews
conf.extensions = ['sphinx.ext.todo', 'sphinx.ext.viewcode', 'autoapi.extension']

```

```
conf.autoapi_type = python
conf.autoapi_dirs = ['.']
conf.templates_path = ['_templates']
conf.exclude_patterns = ['_build', 'Thumbs.db', '.DS_Store']
conf.html_theme = alabaster
conf.html_static_path = ['_static']
conf.latex_engine = lualatex
conf.latex_elements
conf.latex_show_urls = footnote
```

Indices and tables

- `genindex`
- `modindex`
- `search`

c

`conf`, [15](#)

d

`dissociation_curve`, [14](#)

e

`evaluator`, [10](#)

h

`hamiltonian`, [7](#)

m

`mapping`, [2](#)

p

`pauli_string`, [3](#)

s

`sim_no_noise`, [13](#)

`sim_noisy`, [12](#)

`sim_noisy_layout_opt`, [13](#)

`sim_noisy_measfilt_layout_opt`, [15](#)

`solver`, [1](#)

Non-alphabetical

`__add__()` (*pauli_string.LinearCombinaisonPauliString method*), 5
`__getitem__()` (*pauli_string.LinearCombinaisonPauliString method*), 5
`__len__()` (*pauli_string.LinearCombinaisonPauliString method*), 5
`__len__()` (*pauli_string.PauliString method*), 3
`__mul__()` (*pauli_string.LinearCombinaisonPauliString method*), 5
`__mul__()` (*pauli_string.PauliString method*), 3
`__rmul__()` (*pauli_string.LinearCombinaisonPauliString method*), 5
`__rmul__()` (*pauli_string.PauliString method*), 4
`__str__()` (*hamiltonian.FermionicHamiltonian method*), 7
`__str__()` (*pauli_string.LinearCombinaisonPauliString method*), 5
`__str__()` (*pauli_string.PauliString method*), 3

A

`a` (*in module dissociation_curve*), 14
`a` (*in module sim_no_noise*), 13
`a` (*in module sim_noisy*), 12
`a` (*in module sim_noisy_layout_opt*), 13
`a` (*in module sim_noisy_measfilt_layout_opt*), 15
`add_pauli_string_linear_combinaison()`
 (*pauli_string.LinearCombinaisonPauliString method*), 5
`apply_threshold()` (*pauli_string.LinearCombinaisonPauliString method*), 6
`author` (*in module conf*), 15
`autoapi_dirs` (*in module conf*), 16
`autoapi_type` (*in module conf*), 15

B

`BasicEvaluator` (*class in evaluator*), 11
`bitwise_clique_circuit_and_interpreter()`
 (*evaluator.BitwiseCommutingCliqueEvaluator static method*), 12
`BitwiseCommutingCliqueEvaluator` (*class in evaluator*), 12
`bogota` (*in module dissociation_curve*), 14
`bogota` (*in module sim_noisy*), 12
`bogota` (*in module sim_noisy_layout_opt*), 13
`bogota` (*in module sim_noisy_measfilt_layout_opt*), 15
`bogota_conf` (*in module dissociation_curve*), 14
`bogota_conf` (*in module sim_noisy*), 12
`bogota_conf` (*in module sim_noisy_layout_opt*), 13
`bogota_conf` (*in module sim_noisy_measfilt_layout_opt*), 15
`bogota_nm` (*in module dissociation_curve*), 14
`bogota_nm` (*in module sim_noisy*), 12
`bogota_nm` (*in module sim_noisy_layout_opt*), 13

`bogota_nm` (*in module sim_noisy_measfilt_layout_opt*), 15
`bogota_prop` (*in module dissociation_curve*), 14
`bogota_prop` (*in module sim_noisy*), 12
`bogota_prop` (*in module sim_noisy_layout_opt*), 13
`bogota_prop` (*in module sim_noisy_measfilt_layout_opt*), 15

C

`calibration_layout` (*in module dissociation_curve*), 14
`calibration_layout` (*in module sim_noisy_measfilt_layout_opt*), 15
`change_basis()` (*hamiltonian.MolecularFermionicHamiltonian method*), 9
`change_basis()` (*hamiltonian.OneBodyFermionicHamiltonian method*), 8
`change_basis()` (*hamiltonian.TwoBodyFermionicHamiltonian method*), 8
`combine()` (*pauli_string.LinearCombinaisonPauliString method*), 6
`conf`
 module, 15
`copy()` (*pauli_string.PauliString method*), 4
`copyright` (*in module conf*), 15
`counts2array()` (*evaluator.Evaluator method*), 10

D

`dissociation_curve`
 module, 14
`divide_in_bitwise_commuting_cliques()`
 (*pauli_string.LinearCombinaisonPauliString method*), 6

E

`eig()` (*solver.ExactSolver method*), 1
`eval()` (*evaluator.Evaluator method*), 10
`evaluator`
 module, 10
`Evaluator` (*class in evaluator*), 10
`ExactSolver` (*class in solver*), 1
`exclude_patterns` (*in module conf*), 16
`extensions` (*in module conf*), 15

F

`fermionic_hamiltonian_to_linear_combinaison_pauli_string()`
 (*mapping.Mapping method*), 2
`fermionic_operator_linear_combinaison_pauli_string()`
 (*mapping.JordanWigner method*), 2

fermionic_operator_linear_combinaison_pauli_string(*measurable_eigenvalues()* (*evaluator.Evaluator static method*),
(*mapping.Parity method*), 3
FermionicHamiltonian (*class in hamiltonian*), 7
from_integrals() (*hamiltonian.MolecularFermionicHamiltonian*
class method), 8
from_pyscf_mol() (*hamiltonian.MolecularFermionicHamiltonian*
class method), 8
from_str() (*pauli_string.PauliString class method*), 4
from_zx_bits() (*pauli_string.PauliString class method*), 4

G

get_energies() (*in module dissociation_curve*), 14
get_energies() (*in module sim_no_noise*), 14
get_energies() (*in module sim_noisy*), 13
get_energies() (*in module sim_noisy_layout_opt*), 13
get_energies() (*in module sim_noisy_measfilt_layout_opt*), 15
get_integrals() (*hamiltonian.FermionicHamiltonian method*), 7
get_integrals() (*hamiltonian.MolecularFermionicHamiltonian*
method), 9

H

hamiltonian
 module, 7
html_static_path (*in module conf*), 16
html_theme (*in module conf*), 16

I

ids() (*pauli_string.LinearCombinaisonPauliString method*), 6
ids() (*pauli_string.PauliString method*), 4
include_spin() (*hamiltonian.FermionicHamiltonian method*), 7
include_spin() (*hamiltonian.MolecularFermionicHamiltonian*
method), 9
interpret_count_array() (*evaluator.Evaluator static method*), 11
interpret_count_arrays() (*evaluator.Evaluator method*), 10

J

JordanWigner (*class in mapping*), 2

L

latex_elements (*in module conf*), 16
latex_engine (*in module conf*), 16
latex_show_urls (*in module conf*), 16
lcps_h2 (*in module sim_no_noise*), 14
lcps_h2 (*in module sim_noisy*), 12
lcps_h2 (*in module sim_noisy_layout_opt*), 13
lcps_h2 (*in module sim_noisy_measfilt_layout_opt*), 15
LCPSSolver (*class in solver*), 1
LinearCombinaisonPauliString (*class in pauli_string*), 5
lowest_eig_value() (*solver.ExactSolver method*), 1
lowest_eig_value() (*solver.VQESolver method*), 2

M

mapping
 module, 2
Mapping (*class in mapping*), 2
mapping (*in module sim_no_noise*), 14
mapping (*in module sim_noisy*), 12
mapping (*in module sim_noisy_layout_opt*), 13
mapping (*in module sim_noisy_measfilt_layout_opt*), 15
meas_filter (*in module dissociation_curve*), 14
meas_filter (*in module sim_noisy_measfilt_layout_opt*), 15
meas_fitter (*in module dissociation_curve*), 14
meas_fitter (*in module sim_noisy_measfilt_layout_opt*), 15

minimizer (*in module dissociation_curve*), 14
minimizer (*in module sim_no_noise*), 14
minimizer (*in module sim_noisy*), 12
minimizer (*in module sim_noisy_layout_opt*), 13
minimizer (*in module sim_noisy_measfilt_layout_opt*), 15
module
 conf, 15
 dissociation_curve, 14
 evaluator, 10
 hamiltonian, 7
 mapping, 2
 pauli_string, 3
 sim_no_noise, 13
 sim_noisy, 12
 sim_noisy_layout_opt, 13
 sim_noisy_measfilt_layout_opt, 15
 solver, 1
molecular_hamiltonian (*in module sim_no_noise*), 13
molecular_hamiltonian (*in module sim_noisy*), 12
molecular_hamiltonian (*in module sim_noisy_layout_opt*), 13
molecular_hamiltonian (*in module*
 sim_noisy_measfilt_layout_opt), 15
MolecularFermionicHamiltonian (*class in hamiltonian*), 8
mul_coef() (*pauli_string.LinearCombinaisonPauliString method*), 6
mul_coef() (*pauli_string.PauliString method*), 4
mul_linear_combinaison_pauli_string()
 (*pauli_string.LinearCombinaisonPauliString method*), 5
mul_pauli_string() (*pauli_string.PauliString method*), 4

N

N (*in module sim_no_noise*), 14
N (*in module sim_noisy*), 13
N (*in module sim_noisy_layout_opt*), 13
N (*in module sim_noisy_measfilt_layout_opt*), 15
number_of_orbitals() (*hamiltonian.FermionicHamiltonian*
method), 7
number_of_orbitals()
 (*hamiltonian.MolecularFermionicHamiltonian method*), 9

O

OneBodyFermionicHamiltonian (*class in hamiltonian*), 7
out (*in module sim_no_noise*), 13
out (*in module sim_noisy*), 12
out (*in module sim_noisy_layout_opt*), 13
out (*in module sim_noisy_measfilt_layout_opt*), 15

P

Parity (*class in mapping*), 3
pauli_string
 module, 3
pauli_string_based_measurement() (*evaluator.Evaluator*
 static method), 11
pauli_string_circuit_and_interpreter()
 (*evaluator.BasicEvaluator static method*), 11
PauliString (*class in pauli_string*), 3
prepare_eval_circuits() (*evaluator.Evaluator method*), 10
prepare_measurement_circuits_and_interpreters()
 (*evaluator.BasicEvaluator static method*), 11
prepare_measurement_circuits_and_interpreters()
 (*evaluator.BitwiseCommutingCliqueEvaluator static method*),
 12
project (*in module conf*), 15
provider (*in module dissociation_curve*), 14
provider (*in module sim_noisy*), 12

provider (in module *sim_noisy_layout_opt*), 13
 provider (in module *sim_noisy_measfilt_layout_opt*), 15

Q

qasm_simulator (in module *dissociation_curve*), 14
 qasm_simulator (in module *sim_no_noise*), 14
 qasm_simulator (in module *sim_noisy*), 12
 qasm_simulator (in module *sim_noisy_layout_opt*), 13
 qasm_simulator (in module *sim_noisy_measfilt_layout_opt*), 15
 qr (in module *dissociation_curve*), 14
 qr (in module *sim_noisy_measfilt_layout_opt*), 15
 qubit_list (in module *dissociation_curve*), 14
 qubit_list (in module *sim_noisy_measfilt_layout_opt*), 15

R

result (in module *dissociation_curve*), 14
 result (in module *sim_noisy_measfilt_layout_opt*), 15

S

set_linear_combinaison_pauli_string()
 (enumerator.Evaluator method), 10
 shots (in module *dissociation_curve*), 14
 sim_no_noise
 module, 13
 sim_noisy
 module, 12
 sim_noisy_layout_opt
 module, 13
 sim_noisy_measfilt_layout_opt
 module, 15
 solver
 module, 1
 sort() (pauli_string.LinearCombinaisonPauliString method), 6
 spin_tensor (hamiltonian.OneBodyFermionicHamiltonian attribute),
 8
 spin_tensor (hamiltonian.TwoBodyFermionicHamiltonian attribute),
 8

T

templates_path (in module *conf*), 16
 to_linear_combinaison_pauli_string()
 (hamiltonian.MolecularFermionicHamiltonian method), 9
 to_linear_combinaison_pauli_string()
 (hamiltonian.OneBodyFermionicHamiltonian method), 8
 to_linear_combinaison_pauli_string()
 (hamiltonian.TwoBodyFermionicHamiltonian method), 8
 to_matrix() (pauli_string.LinearCombinaisonPauliString method), 6
 to_matrix() (pauli_string.PauliString method), 5
 to_xz_bits() (pauli_string.LinearCombinaisonPauliString method), 6
 to_xz_bits() (pauli_string.PauliString method), 4
 to_zx_bits() (pauli_string.LinearCombinaisonPauliString method), 6
 to_zx_bits() (pauli_string.PauliString method), 4
 TwoBodyFermionicHamiltonian (class in hamiltonian), 8

V

varform_4qubits_1param (in module *dissociation_curve*), 14
 varform_4qubits_1param (in module *sim_no_noise*), 13
 varform_4qubits_1param (in module *sim_noisy*), 12
 varform_4qubits_1param (in module *sim_noisy_layout_opt*), 13
 varform_4qubits_1param (in module
 sim_noisy_measfilt_layout_opt), 15
 VQESolver (class in solver), 2