ueg_fci Documentation

Release 0.1

James Spencer

March 20, 2016

Contents

Full Configuration Interaction (FCI) method for simple quantum systems. Currently the uniform electron gas and a spinless lattice model are implemented.

This is a particularly naive implementation in python: little attempt is made to conserve memory or CPU time. Nevertheless, it is useful for small test calculations, in particular for investigating ideas about the sign problem in the Full Configuration Interaction Quantum Monte Carlo (FCIQMC) method discussed by Spencer, Blunt and Foulkes (J. Chem. Phys. 136, 054110 (2012); arXiv:1110.5479).

Requires python 2.6 or later and numpy. All code is compatible with python 3.

CHAPTER 1

hamil

Base classes and functions for generating and interacting with Hamiltonian matrices.

class hamil.**Hamiltonian**(*sys*, *basis*, *tau=0.1*)

Base Hamiltonian class.

The relevant subclass which provides the appropriate matrix elements should be used.

Parameters

- **sys** – object describing system to be studied; used only in the virtual matrix element functions
- **basis** (*iterable of iterables of single-particle basis functions*) – set of many-particle basis functions
- **tau** (*float*) – timestep by which to propogate psip distributions in imaginary time

---

Note: This is a base class; basis and sys must be appropriate to the actual subclass used and are specific to the required system and underlying many-particle basis set.

---

sys = None
system to be studied

basis = None
set of many-particle basis functions

nbasis = None
number of many-particle basis functions (i.e. length basis)

hamil = None
Hamiltonian matrix in the basis set of the many-particle basis functions

tau = None
timestep by which to propogate psip distributions in imaginary time

pos_propogator = None
positive propogator matrix, T^+ where $T = 1 - \tau H = T^+ - T^-$ and T^+ elements are non-negative.

neg_propogator = None
negative propogator matrix, T^- where $T = 1 - \tau H = T^+ - T^-$ and T^- elements are non-negative.

mat_fn_diag(*b*)
Calculate a diagonal Hamiltonian matrix element.

Warning: Virtual member. Must be appropriately implemented in a subclass.

Parameters **b** (*iterable of single-particle basis functions*) – a many-particle basis function, $|b\rangle$

Return type float

Returns $\langle b|H|b\rangle$.

mat_fn_offdiag(*b1*, *b2*)
Calculate an off-diagonal Hamiltonian matrix element.

Warning: Virtual member. Must be appropriately implemented in a subclass.

Parameters

- **b1** (*iterable of single-particle basis functions*) – a many-particle basis function, $|b_1\rangle$
- **b2** (*iterable of single-particle basis functions*) – a many-particle basis function, $|b_2\rangle$

Return type float

Returns $\langle b_1|H|b_2\rangle$.

eigh()

Returns (eigenvalues, eigenvectors) of the Hamiltonian matrix.

eigvalsh()

Returns eigenvalues of the Hamiltonian matrix.

negabs_off_diagonal_elements()
Set off-diagonal elements of the Hamiltonian matrix to be negative.

This converts the Hamiltonian into the lesser sign-problem matrix discussed by Spencer, Blunt and Foulkes.

negabs_diagonal_elements()
Set diagonal elements of the Hamiltonian matrix to be negative.

This, when called after negabs_offdiagonal_elements, converts the Hamiltonian into the greater sign-problem matrix discussed by Spencer, Blunt and Foulkes.

propogate(*pos_psips*, *neg_psips*)
Propogates a psip (psi-particle) distribution for a single timestep.

Parameters

- **pos_psips** (*1D array or list (length nbasis)*) – distribution of positive psips at time $t = n\tau$ on the many-fermion basis set.
- **neg_psips** (*1D array or list (length nbasis)*) – distribution of negative psips at time $t = n\tau$ on the many-fermion basis set.

Returns (next_pos_psips, next_neg_psips) — positive and negative psip distributions at time $t = (n + 1)\tau$.

hamil.**hartree_excitation**(*p1*, *p2*)
Find the excitation connecting two Hartree products.

Parameters

- **p1** (*iterable of single-particle basis functions*) – a Hartree product basis function

- **p2** (*iterable of single-particle basis functions*) – a many-particle basis function

Returns

(from_1, to_1) where:

from_1 list of single-particle basis functions excited from p1

to_2 list of single-particle basis functions excited into p2

hamil.**determinant_excitation**(*d1*, *d2*)

Find the excitation connecting two Slater determinants.

Parameters

- **p1** (*iterable of single-particle basis functions*) – a Slater determinant basis function
- **p2** (*iterable of single-particle basis functions*) – a Slater determinant basis function

Returns

(from_1, to_1, nperm) where:

from_1 list of single-particle basis functions excited from d1

to_2 list of single-particle basis functions excited into d2

nperm number of permutations required to align the two determinants such that the orders of single-particle basis functions are in maximum agreement. This, in general, is not the minimal possible number of permutations but the parity of the permutations, which is all that is required for calculating matrix elements, is correct.

hamil.**permanent_excitation**(*p1*, *p2*)

Find the excitation connecting two permanents.

Parameters

- **p1** (*iterable of single-particle basis functions*) – a permanent basis function
- **p2** (*iterable of single-particle basis functions*) – a permanent basis function

Returns

(from_1, to_1) where:

from_1 list of single-particle basis functions excited from p1

to_2 list of single-particle basis functions excited into p2

CHAPTER 2

lattice_fci

Implement the spinless fermion lattice model defined by the Hamiltonian

$$H = -\sum_{<ij>} t\left(c_j^\dagger c_i + c_i^\dagger c_j - u n_i n_j\right)$$

in both Hartree product and Slater determinant bases on a square lattice of dimension $L \times L$.

This is a simple example of when the sign problem in FCIQMC is identical in both first- and second-quantized bases.

This is an implementation of the model originally proposed by Michael Kolodrubetz and Bryan Clark (Princeton).

class lattice_fci.**LatticeSite**(*x*, *y*, *L*)

Basis function of a lattice site at location (x, y) in a $L \times L$ square lattice.

Parameters

- **x** (*integer*) – x coordinate of lattice site.
- **y** (*integer*) – y coordinate of lattice site.
- **L** (*integer*) – dimension of lattice cell.

x = None

x coordinate of lattice site.

y = None

y coordinate of lattice site.

loc = None

unique index of lattice site.

class lattice_fci.**LatticeModel**(*nfermions*, *L*, *t=1*, *u=1*)

Representation of the lattice model system defined by the Hamiltonian:

$$H = -\sum_{<ij>} t\left(c_j^\dagger c_i + c_i^\dagger c_j - u n_i n_j\right)$$

Parameters

- **nfermions** (*integer*) – dimension of simulation cell
- **L** (*integer*) – number of spinless fermions in the simulation cell
- **t** (*float*) – hopping (kinetic) parameter in Hamiltonian operator
- **u** (*float*) – Coulomb parameter in Hamiltonian operator

nfermions = None

number of spinless fermions in the simulation cell

L = None
dimension of simulation cell

t = None
Hopping parameter

u = None
Coulomb parameter

hopping_int(*site1*, *site2*)
Calculate the hopping integral between two fermions.

Parameters

- **site1** (*LatticeSite*) – lattice site, s_1, occupied by a fermion
- **site2** (*LatticeSite*) – lattice site, s_2, occupied by a fermion

Returns $u \sum_{<ij>} \langle s_1 | u n_i n_j | s_2 \rangle$

coulomb_int(*site1*, *site2*)
Calculate the Coulomb integral between two fermions.

Parameters

- **site1** (*LatticeSite*) – lattice site, s_1, occupied by a fermion
- **site2** (*LatticeSite*) – lattice site, s_2, occupied by a fermion

Returns $-t \sum_{<ij>} \langle s_1 | c_j^\dagger c_i + c_i^\dagger c_j | s_2 \rangle$

lattice_fci.**init_lattice_basis**(*nfermions*, *L*)
Construct many-fermion bases.

Parameters

- **nfermions** (*integer*) – number of fermions in a simulation cell.
- **L** (*integer*) – dimension of 2D square simulation cell, where each lattice site contains a single-partle basis function.

Returns

(hartree_producs, determinants) where:

hartree_products tuple containing all Hartree products.

determinants tuple containing all Slater determinants.

Determinants and Hartree products are represented as tuples of *LatticeSite* objects.

class lattice_fci.**HartreeLatticeHamiltonian**(*sys*, *basis*, *tau=0.1*)
Bases: *hamil.Hamiltonian*

Hamiltonian for the fermion lattice model in a Hartree product basis.

sys must be a *LatticeModel* object and the underlying single-particle basis functions must be *LatticeSite* objects.

The Hartree product basis is the set of all possible permutations of fermions in the single-particle basis set.

mat_fn_diag(*b*)
Calculate a diagonal Hamiltonian matrix element.

Parameters **b** (iterable of *LatticeSite* objects) – a Hartree product basis function, $|b\rangle$

Return type float

Returns $\langle b|H|b\rangle$

mat_fn_offdiag(*bi*, *bj*)
Calculate an off-diagonal Hamiltonian matrix element.

Parameters

- **bi** (iterable of *LatticeSite* objects) – a Hartree product basis function, $|b_i\rangle$
- **bj** (iterable of *LatticeSite* objects) – a Hartree product basis function, $|b_j\rangle$

Return type float

Returns $\langle b_i|H|b_j\rangle$.

class lattice_fci.**DeterminantLatticeHamiltonian**(*sys*, *basis*, *tau=0.1*)
Bases: *hamil.Hamiltonian*

Hamiltonian for the fermion lattice model in a Slater determinant basis.

sys must be a *LatticeModel* object and the underlying single-particle basis functions must be *LatticeSite* objects.

The Slater determinant basis is the set of all possible combinations of fermions in the single-particle basis set.

mat_fn_diag(*b*)
Calculate a diagonal Hamiltonian matrix element.

Parameters **b** (iterable of *LatticeSite* objects) – a Slater determinant basis function, $|b\rangle$

Return type float

Returns $\langle b|H|b\rangle$

mat_fn_offdiag(*bi*, *bj*)
Calculate an off-diagonal Hamiltonian matrix element.

Parameters

- **bi** (iterable of *LatticeSite* objects) – a Slater determinant basis function, $|b_i\rangle$
- **bj** (iterable of *LatticeSite* objects) – a Slater determinant basis function, $|b_j\rangle$

Return type float

Returns $\langle b_i|H|b_j\rangle$.

lattice_fci.**print_wfn**(*basis*, *pos*, *neg*)
Print out a stochastic wavefunction represented on a basis.

Parameters

- **basis** (iterable of iterable of *LatticeSite* objects) – many-fermion basis
- **pos** (*1D vector of length basis*) – weight of positive psips on each basis function
- **neg** (*1D vector of length basis*) – weight of negative psips on each basis function

lattice_fci.**print_two_fermion_wfn**(*basis*, *pos*, *neg*, *L*)
Print out a stochastic wavefunction represented on a basis.

Assumes there are two fermions in the simulation cell.

Parameters

- **basis** (iterable of iterable of *LatticeSite* objects) – many-fermion basis

- **pos** (*1D vector of length basis*) – weight of positive psips on each basis function
- **neg** (*1D vector of length basis*) – weight of negative psips on each basis function
- **L** (*integer*) – dimension of lattice cell.

CHAPTER 3

ueg_fci

Full Configuration Interaction (FCI) method for the uniform electron gas.

This is a particularly simple implementation: little attempt is made to conserve memory or CPU time. Nevertheless, it is useful for small test calculations, in particular for investigating ideas about the sign problem in the Full Configuration Interaction Quantum Monte Carlo method discussed by Spencer, Blunt and Foulkes (J. Chem. Phys. 136, 054110 (2012); arXiv:1110.5479).

FCI calculations can be performed in a Slater determinant, permanent or Hartree product basis.

Note that no attempt is made to tackle finite size effects.

Warning: The Hartree product basis set is much larger than the determinant/permanent basis set, even for tiny systems. Thus the system sizes which can be tackled using the Hartree product basis are far more limited.

class ueg_fci.**BasisFn**(*i*, *j*, *k*, *L*, *spin*)
Basis function with wavevector $2\pi(i, j, k)^T/L$ of the desired spin.

Parameters

- **i, j, k** (*integer*) – integer labels (quantum numbers) of the wavevector
- **L** (*float*) – dimension of the cubic simulation cell of size $L \times L \times L$
- **spin** (*integer*) – spin of the basis function (-1 for a down electron, +1 for an up electron)

ueg_fci.**total_momentum**(*basis_iterable*)
Calculate the total momentum of a many-particle basis function.

Parameters **basis_iterable** (iterable of *BasisFn* objects) – many-particle basis function

Returns the total momentum, in units of $2\pi/L$, of the basis functions in basis_iterable

class ueg_fci.**UEG**(*nel*, *nalpha*, *nbeta*, *rs*)
Create a representation of a 3D uniform electron gas.

Parameters

- **nel** (*integer*) – number of electrons
- **nalpha** (*integer*) – number of alpha (spin-up) electrons
- **nbeta** (*integer*) – number of beta (spin-down) electrons
- **rs** (*float*) – electronic density

nel = None
number of electrons

nalpha = None
number of alpha (spin-up) electrons

nbeta = None
number of beta (spin-down) electrons

rs = None
electronic density

L = None
length of the cubic simulation cell containing nel electrons at the density of rs

Omega = None
volume of the cubic simulation cell containing nel electrons at the density of rs

coulomb_int(*q*)
Calculate the Coulomb integral $\langle k\ k'|k+q\ k'-q\rangle$.

The Coulomb integral:

$$\langle k\ k'|k+q\ k'-q\rangle = \frac{4\pi}{\Omega q^2}$$

where Ω is the volume of the simulation cell, is independent of the wavevectors k and k' and hence only the q vector is required.

Parameters **q** (*numpy.array*) – momentum transfer vector (in absolute units)

ueg_fci.**init_ueg_basis**(*sys*, *cutoff*, *sym*)
Create single-particle and the many-particle bases.

Parameters

- **sys** (*UEG*) – UEG system to be studied.
- **cutoff** (*float*) – energy cutoff, in units of $(2\pi/L)^2$, defining the single-particle basis. Only single-particle basis functions with a kinetic energy equal to or less than the cutoff are considered.
- **sym** (*numpy.array*) – integer vector defining the wavevector, in units of $2\pi/L$, representing the desired symmetry. Only Hartree products and determinants of this symmetry are returned.

Returns

(basis_fns, hartree_products, determinants) where:

basis_fns tuple of relevant *BasisFn* objects, ie the single-particle basis set.

hartree_products tuple containing all Hartree products formed from basis_fns.

determinants tuple containing all Slater determinants formed from basis_fns.

Determinants and Hartree products are represented as tuples of *BasisFn* objects.

class ueg_fci.**HartreeUEGHamiltonian**(*sys*, *basis*, *tau=0.1*)
Bases: *hamil.Hamiltonian*

Hamiltonian class for the UEG in a Hartree product basis.

sys must be a *UEG* object and the single-particle basis functions must be *BasisFn* objects.

The Hartree product basis is the set of all possible permutations of electrons in the single-particle basis set. It is sufficient (and cheaper) to consider one spin and momentum block of the Hamiltonian at a time.

mat_fn_diag(*p*)
Calculate a diagonal Hamiltonian matrix element.

Parameters **p** (iterable of *BasisFn* objects) – a Hartree product basis function, $|p_1\rangle$

Return type float

Returns $\langle p|H|p\rangle$

mat_fn_offdiag(*p1*, *p2*)
Calculate an off-diagonal matrix element.

Parameters

- **p1** (iterable of *BasisFn* objects) – a Hartree product basis function, $|p_1\rangle$
- **p2** (iterable of *BasisFn* objects) – a Hartree product basis function, $|p_2\rangle$

Return type float

Returns $\langle p_1|H|p_2\rangle$

class ueg_fci.**DeterminantUEGHamiltonian**(*sys*, *basis*, *tau=0.1*)
Bases: *hamil.Hamiltonian*

Hamiltonian class for the UEG in a Slater determinant basis.

sys must be a *UEG* object and the single-particle basis functions must be *BasisFn* objects.

The Slater determinant basis is the set of all possible combinations of electrons in the single-particle basis set. It is sufficient (and cheaper) to consider one spin and momentum block of the Hamiltonian at a time.

mat_fn_diag(*d*)
Calculate a diagonal Hamiltonian matrix element.

Parameters **d** (iterable of *BasisFn* objects) – a Slater determinant basis function, $|d\rangle$

Return type float

Returns $\langle d|H|d\rangle$

mat_fn_offdiag(*d1*, *d2*)
Calculate an off-diagonal Hamiltonian matrix element.

Parameters

- **d1** (iterable of *BasisFn* objects) – a Slater determinant basis function, $|d_1\rangle$
- **d2** (iterable of *BasisFn* objects) – a Slater determinant basis function, $|d_2\rangle$

Return type float

Returns $\langle d_1|H|d_2\rangle$.

class ueg_fci.**PermanentUEGHamiltonian**(*sys*, *basis*, *tau=0.1*)
Bases: *hamil.Hamiltonian*

Hamiltonian class for the UEG in a permanent basis.

sys must be a *UEG* object and the single-particle basis functions must be *BasisFn* objects.

The permanent basis is the set of all possible combinations of electrons in the single-particle basis set, and hence is identical to the Slater determinant basis. It is sufficient (and cheaper) to consider one spin and momentum block of the Hamiltonian at a time.

mat_fn_diag(*p*)
Calculate a diagonal Hamiltonian matrix element.

Parameters **p** (iterable of *BasisFn* objects) – a permanent basis function, $|p\rangle$

Return type float

Returns $\langle p|H|p\rangle$.

mat_fn_offdiag(*p1*, *p2*)
Calculate an off-diagonal Hamiltonian matrix element.

Parameters

- **p1** (iterable of *BasisFn* objects) – a permanent basis function, $|p1\rangle$
- **p2** (iterable of *BasisFn* objects) – a permanent basis function, $|p2\rangle$

Return type float

Returns $\langle p_1|H|p_2\rangle$.

CHAPTER 4

lattice_propogation

Script which uses the FCIQMC algorithm without annihilation to propogate positive and negative psips using Hamiltonians defined in both Hartree product and Slater determinant bases for two spinless fermions on a 4×4 lattice with periodic boundary conditions. The Hamiltonian operator is:

$$H = -\sum_{<ij>} t \left(c_j^\dagger c_i + c_i^\dagger c_j - u n_i n_j \right).$$

This is an example where the evolution of an FCIQMC calculation is step-by-step identical in both first- and second-quantized basis sets.

This is an independent implementation of work originally performed by Michael Kolodrubetz and Bryan Clark (Princeton).

CHAPTER 5

ueg_sign_problem

Script to investigate the sign problem in a UEG system in various bases.

Execute to calculate and print out the lowest eigenvalues of Hamiltonian matrices (and those related to the FCIQMC sign problem) for a small UEG system using the ueg_fci module.

ueg_sign_problem.**print_title**(*title*, *under='='*)
Print the underlined title.

Parameters

- **title** (*string*) – section title to print out
- **under** (*string*) – single character used to underline the title

ueg_sign_problem.**worker**(*label*, *sys*, *basis*, *Hamil*, *nprint*)
Helper function to construct and diagonalise matrices related to the sign problem.

Parameters

- **label** (*string*) – label of the many-particle basis function used
- **sys** (*ueg_fci.UEG*) – desired UEG system; passed to Hamil
- **basis** (iterable of iterables of *ueg_fci.BasisFn*) – set of many-particle basis functions
- **Hamil** (ueg_fci.UEGHamiltonian subclass) – appropriate Hamiltonian for the basis provided
- **nprint** (*integer*) – number of eigenvalues to print out

CHAPTER 6

License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 7

Indices and tables

- genindex
- search

h

l

u

B

C

D

E

H

I

L

M

N

O

P

R

S

T

U

W

X

Y