Hamiltonian Time Evolution and Expectation Value Computation

Run in Google Colab (https://colab.research.google.com/github/quantumlib/OpenFermion/blob/master/docs/

This tutorial describes the FQE's capabilities for Hamiltonian time-evolution and expectation value estimation

Where possible, LiH will be used as an example molecule for the API.

```
try:
    import fqe
except ImportError:
    !pip install fqe --quiet
```

```
Print = True
from openfermion import FermionOperator, MolecularData
from openfermion.utils import hermitian_conjugated
import numpy
import fqe

numpy.set_printoptions(floatmode='fixed', precision=6, linewidth=80, suppress=
numpy.random.seed(seed=409)
```

```
$ curl -O https://raw.githubusercontent.com/quantumlib/OpenFermion-FQE/master/
```

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 62086  100 62086    0     0   259k      0 --:--:-- --:--:-- --:--:--  260k
```

```
import build_lih_data



h1e, h2e, wfn = build_lih_data.build_lih_data('energy')
lih_hamiltonian = fqe.get_restricted_hamiltonian(([h1e, h2e]))
lihwfn = fqe.Wavefunction([[4, 0, 6]])
lihwfn.set_wfn(strategy='from_data', raw_data={(4, 0): wfn})
if Print:
    lihwfn.print_wfn()



Sector N = 4 : S_z = 0
a'000011'b'000011' (-0.9870890035778126+0j)
a'000011'b'000101' (-0.0384854879340124+0j)
a'000011'b'100001' (-0.0036477466809967+0j)
a'000101'b'000011' (-0.0384854879340073+0j)
a'000101'b'000101' (0.0344508540105218+0j)
a'000101'b'100001' (-0.0592953695921639+0j)
a'000101'b'100010' (-0.0010777697106179+0j)
a'001001'b'001001' (0.0271656962509733+0j)
a'001001'b'001010' (-0.0031209939957822+0j)
a'010001'b'010001' (0.0271656962509733+0j)
a'010001'b'010010' (-0.0031209939957822+0j)
a'100001'b'000011' (-0.0036477466809997+0j)
a'100001'b'000101' (-0.0592953695921649+0j)
```

Application of one- and two-body fermionic gates

The API for time propagation can be invoked through the fqe namespace or the wavefunction object

```
# dummy geometry
from openfermion.chem.molecular_data import spinorb_from_spatial
from openfermion import jordan_wigner, get_sparse_operator, InteractionOperato

h1s, h2s = spinorb_from_spatial(h1e, numpy.einsum("ijlk", -2 * h2e) * 0.5)
mol = InteractionOperator(0, h1s, h2s)
ham_fop = get_fermion_operator(mol)
ham_mat = get_sparse_operator(jordan_wigner(ham_fop)).toarray()
```

```
from scipy.linalg import expm
time = 0.01
evolved1 = lihwfn.time_evolve(time, lih_hamiltonian)
if Print:
    evolved1.print_wfn()
evolved2 = fqe.time_evolve(lihwfn, time, lih_hamiltonian)
if Print:
    evolved2.print_wfn()
assert numpy.isclose(fqe.vdot(evolved1, evolved2), 1)
cirq_wf = fqe.to_cirq(lihwfn)
evolve_cirq = expm(-1j * time * ham_mat) @ cirq_wf
test_evolve = fqe.from_cirq(evolve_cirq, thresh=1.0E-12)
assert numpy.isclose(fqe.vdot(test_evolve, evolved1), 1)
```

```
Sector N = 4 : S_z = 0
a'000011'b'000011' (-0.9832017407384288-0.08751592993809965j)
a'000011'b'000101' (-0.038333927909412015-0.003412147794935153j)
a'000011'b'100001' (-0.003633381455202132-0.0003234115814680900 5j)
a'000101'b'000011' (-0.038333927909406935-0.003412147794934707 4j)
a'000101'b'000101' (0.03431518282571416+0.003054434433325243j)
a'000101'b'100001' (-0.05906185803441181-0.005257164690403508j)
a'000101'b'100010' (-0.001073525344578889-9.555564554363788e-05j)
a'001001'b'001001' (0.02705871481472957+0.002408527635427536 7j)
a'001001'b'001010' (-0.003108703183821363-0.000267709302556 57j)
a'010001'b'010001' (0.02705871481472957+0.002408527635427536 7j)
a'010001'b'010010' (-0.003108703183821363-0.000267709302556 57j)
a'100001'b'000011' (-0.0036333814552024306-0.0003234115814681176j)
a'100001'b'000101' (-0.0590618580344128-0.005257164690403600 5j)
```

Exact evolution implementation of quadratic Hamiltonians

Listed here are examples of evolving the special Hamiltonians.

Diagonal Hamiltonian evolution is supported.

```
wfn = fqe.Wavefunction([[4, 2, 4]])
wfn.set_wfn(strategy='random')
if Print:
    wfn.print_wfn()

diagonal = FermionOperator('0^ 0', -2.0) + \
           FermionOperator('1^ 1', -1.7) + \
```

```
            FermionOperator('2^ 2', -0.7) + \
            FermionOperator('3^ 3', -0.55) + \
            FermionOperator('4^ 4', -0.1) + \
            FermionOperator('5^ 5', -0.06) + \
            FermionOperator('6^ 6', 0.5) + \
            FermionOperator('7^ 7', 0.3)
if Print:
    print(diagonal)

evolved = wfn.time_evolve(time, diagonal)
if Print:
    evolved.print_wfn()
```

```
Sector N = 4 : S_z = 2
a'0111'b'0001' (0.06410384332680892+0.006457824553220213j)
a'0111'b'0010' (-0.04750282105641742-0.004806746363757904j)
a'0111'b'0100' (-0.13678251578172124-0.14669354294155937j)
a'0111'b'1000' (-0.17590908446219902+0.18037032855388507j)
a'1011'b'0001' (-0.024476509055323645-0.35002973813257837j)
a'1011'b'0010' (-0.2930654911928577 4-0.2499369934504 2755j)
a'1011'b'0100' (-0.24463823979331925+0.216230018507552j)
a'1011'b'1000' (0.2630370061724331-0.3219176890144332j)
a'1101'b'0001' (0.06881598452335648-0.11820307831930973j)
a'1101'b'0010' (0.07453400242973146-0.06864240203689581j)
a'1101'b'0100' (-0.10280462830183762-0.20883766019199357j)
a'1101'b'1000' (0.18607595463278417-0.08025697543327445j)
a'1110'b'0001' (0.11456775354687777+0.3103763043669054j)
```

Exact evolution of dense quadratic hamiltonians is supported. Here is an evolution example using a spin restricted Hamiltonian on a number and spin conserving wavefunction

```
norb = 4
h1e = numpy.zeros((norb, norb), dtype=numpy.complex128)
for i in range(norb):
    for j in range(norb):
        h1e[i, j] += (i+j) * 0.02
    h1e[i, i] += i * 2.0

hamil = fqe.get_restricted_hamiltonian((h1e,))
wfn = fqe.Wavefunction([[4, 0, norb]])
wfn.set_wfn(strategy='random')
initial_energy = wfn.expectationValue(hamil)
print('Initial Energy: {}'.format(initial_energy))
evolved = wfn.time_evolve(time, hamil)
final_energy = evolved.expectationValue(hamil)
```

```
print('Final Energy:   {}'.format(final_energy))
```

```
Initial Energy: (12.222344460445168+5.551115123125783e-17j)
Final Energy:   (12.222344460445179+1.9168694409543718e-16j)
```

The GSO Hamiltonian is for evolution of quadratic hamiltonians that are spin broken and number conserving.

```
norb = 4
h1e = numpy.zeros((2*norb, 2*norb), dtype=numpy.complex128)
for i in range(2*norb):
    for j in range(2*norb):
        h1e[i, j] += (i+j) * 0.02
    h1e[i, i] += i * 2.0

hamil = fqe.get_gso_hamiltonian((h1e,))
wfn = fqe.get_number_conserving_wavefunction(4, norb)
wfn.set_wfn(strategy='random')
initial_energy = wfn.expectationValue(hamil)
print('Initial Energy: {}'.format(initial_energy))
evolved = wfn.time_evolve(time, hamil)
final_energy = evolved.expectationValue(hamil)
print('Final Energy:   {}'.format(final_energy))
```

```
Initial Energy: (28.720309503101298-2.393918396847994e-16j)
Final Energy:   (28.72030950310125+4.0939474033052647e-16j)
```

The BCS hamiltonian evovles spin conserved and number broken wavefunctions.

```
norb = 4
time = 0.001
wfn_spin = fqe.get_spin_conserving_wavefunction(2, norb)
hamil = FermionOperator('', 6.0)
for i in range(0, 2*norb, 2):
    for j in range(0, 2*norb, 2):
        opstring = str(i) + ' ' + str(j + 1)
        hamil += FermionOperator(opstring, (i+1 + j*2)*0.1 - (i+1 + 2*(j + 1))
        opstring = str(i) + '^ ' + str(j + 1) + '^ '
        hamil += FermionOperator(opstring, (i+1 + j)*0.1 + (i+1 + j)*0.1j)
```

```
h_noncon = (hamil + hermitian_conjugated(hamil))/2.0
if Print:
    print(h_noncon)

wfn_spin.set_wfn(strategy='random')
if Print:
    wfn_spin.print_wfn()

spin_evolved = wfn_spin.time_evolve(time, h_noncon)
if Print:
    spin_evolved.print_wfn()
```

```
6.0 [] +
(0.05-0.15000000000000002j) [0 1] +
(0.25-0.35000000000000003j) [0 3] +
(0.45-0.55j) [0 5] +
(0.65-0.75j) [0 7] +
(0.05+0.05j) [0^ 1^] +
(0.15000000000000002+0.15000000000000002j) [0^ 3^] +
(0.25+0.25j) [0^ 5^] +
(0.35000000000000003+0.35000000000000003j) [0^ 7^] +
(0.05-0.05j) [1 0] +
(0.15000000000000002-0.15000000000000002j) [1 2] +
(0.25-0.25j) [1 4] +
(0.35000000000000003-0.35000000000000003j) [1 6] +
(0.05+0.15000000000000002j) [1^ 0^] +
```

Exact Evolution Implementation of Diagonal Coulomb terms

```
norb = 4
wfn = fqe.Wavefunction([[5, 1, norb]])
vij = numpy.zeros((norb, norb, norb, norb), dtype=numpy.complex128)
for i in range(norb):
            for j in range(norb):
                vij[i, j] += 4*(i % norb + 1)*(j % norb + 1)*0.21

wfn.set_wfn(strategy='random')

if Print:
    wfn.print_wfn()

hamil = fqe.get_diagonalcoulomb_hamiltonian(vij)

evolved = wfn.time_evolve(time, hamil)
if Print:
```

```
    evolved.print_wfn()
```

```
Sector N = 5 : S_z = 1
a'0111'b'0011' (0.176719763323888+0.07705027591386174j)
a'0111'b'0101' (0.2651161627120653+0.008961114965964636j)
a'0111'b'1001' (-0.21500788967807205+0.18883222357115242j)
a'0111'b'0110' (0.05899888340530983+0.24718508330513655j)
a'0111'b'1010' (0.026779941886877087+0.178826903379466j)
a'0111'b'1100' (0.040105194632471065-0.13765308148589672j)
a'1011'b'0011' (-0.07737195675705617-0.13361350504300393j)
a'1011'b'0101' (-0.13687313057404554+0.29187074296331617j)
a'1011'b'1001' (-0.06315193751752252-0.015963783432630548j)
a'1011'b'0110' (0.1412070788641405+0.1547599681431957j)
a'1011'b'1010' (0.19284938811527336+0.05815106367478927j)
a'1011'b'1100' (0.0601422593196422+0.28502769589072846j)
a'1101'b'0011' (-0.18150775903742453+0.016874070450244708j)
```

Exact evolution of individual n-body anti-Hermitian gnerators

```
norb = 3
nele = 4
ops = FermionOperator('5^ 1^ 2 0', 3.0 - 1.j)
ops += FermionOperator('0^ 2^ 1 5', 3.0 + 1.j)
wfn = fqe.get_number_conserving_wavefunction(nele, norb)
wfn.set_wfn(strategy='random')
wfn.normalize()
if Print:
    wfn.print_wfn()
evolved = wfn.time_evolve(time, ops)
if Print:
    evolved.print_wfn()
```

```
Sector N = 4 : S_z = -2
a'001'b'111' (0.15192470437087405+0.002030736551023001j)
a'010'b'111' (-0.03463482137049871+0.27009718718635434j)
a'100'b'111' (-0.227167326143701-0.042680013855770464j)
Sector N = 4 : S_z = 0
a'011'b'011' (-0.3135681471723931+0.02343272674191714j)
a'011'b'101' (-0.02499985483324474+0.08816424695516287j)
a'011'b'110' (0.08135663091791555-0.2354257740286914j)
a'101'b'011' (0.10687726150559686-0.2956005054158534j)
a'101'b'101' (0.17170942273731685+0.06147474695535962j)
a'101'b'110' (-0.06659232012041506-0.17366091158923022j)
```

```
a'110'b'011' (0.1065876871906572+0.05034757634263565j)
a'110'b'101' (-0.5439254382876493-0.12805668259580452j)
```

Approximate evolution of sums of n-body generators

Approximate evolution can be done for dense operators.

```
lih_evolved = lihwfn.apply_generated_unitary(time, 'taylor', lih_hamiltonian,
if Print:
    lih_evolved.print_wfn()
```

```
Sector N = 4 : S_z = 0
a'000011'b'000011' (-0.9870501056639676-0.008762984250848186j)
a'000011'b'000101' (-0.0384839713479148-0.00034165891188627537j)
a'000011'b'100001' (-0.003647602935297334-3.238325409440231e-05j)
a'000101'b'000011' (-0.0384839713479097-0.00034165891188623075j)
a'000101'b'000101' (0.03444949641617426+0.0003058410149547989j)
a'000101'b'100001' (-0.059293032957665366-0.000526400752638063j)
a'000101'b'100010' (-0.0010777272393492372-9.568002262034463e-06j)
a'001001'b'001001' (0.02716462574072948+0.0002411662625463919j)
a'001001'b'001010' (-0.003120871007714629-2.770694732178501e-05j)
a'010001'b'010001' (0.02716462574072948+0.0002411662625463919j)
a'010001'b'010010' (-0.003120871007714629-2.770694732178501e-05j)
a'100001'b'000011' (-0.0036476029352976343-3.238325409440512e-05j)
a'100001'b'000101' (-0.059293032957666365-0.0005264007526380719j)
```

```
norb = 2
nalpha = 1
nbeta = 1
nele = nalpha + nbeta
time = 0.05
h1e = numpy.zeros((norb*2, norb*2), dtype=numpy.complex128)
for i in range(2*norb):
    for j in range(2*norb):
        h1e[i, j] += (i+j) * 0.02
    h1e[i, i] += i * 2.0
hamil = fqe.get_general_hamiltonian((h1e,))
spec_lim = [-1.13199078e-03, 6.12720338e+00]
wfn = fqe.Wavefunction([[nele, nalpha - nbeta, norb]])
wfn.set_wfn(strategy='random')
if Print:
    wfn.print_wfn()
evol_wfn = wfn.apply_generated_unitary(time, 'chebyshev', hamil, spec_lim=spec
if Print:
```

```
    evol_wfn.print_wfn()
```

```
Sector N = 2 : S_z = 0
a'01'b'01' (0.3495794429313121+0.022158760002413482j)
a'01'b'10' (0.5664453364412798-0.3945979322462498j)
a'10'b'01' (-0.39816876837018494+0.14573910133744658j)
a'10'b'10' (-0.4662060110250305-0.06007542464228622j)
Sector N = 2 : S_z = 0
a'01'b'01' (0.344436410676463-0.05101455179951593j)
a'01'b'10' (0.4210294240976466-0.548162567744529j)
a'10'b'01' (-0.33530047561074994+0.26085960635871003j)
a'10'b'10' (-0.4509597513637166+0.13105059079720655j)
```

API for determining desired expectation values

```
rdm1 = lihwfn.expectationValue('i^ j')
if Print:
    print(rdm1)
val = lihwfn.expectationValue('5^ 3')
if Print:
    print(2.*val)
trdm1 = fqe.expectationValue(lih_evolved, 'i j^', lihwfn)
if Print:
    print(trdm1)
val = fqe.expectationValue(lih_evolved, '5 3^', lihwfn)
if Print:
    print(2*val)
```

```
[[ 1.999908+0.000000j -0.000284+0.000000j  0.000441+0.000000j
  -0.000000+0.000000j -0.000000+0.000000j -0.001285+0.000000j]
 [-0.000284+0.000000j  1.951766+0.000000j  0.073757+0.000000j
   0.000000+0.000000j  0.000000+0.000000j  0.010948+0.000000j]
 [ 0.000441+0.000000j  0.073757+0.000000j  0.012402+0.000000j
   0.000000+0.000000j  0.000000+0.000000j -0.017277+0.000000j]
 [-0.000000+0.000000j  0.000000+0.000000j  0.000000+0.000000j
   0.001525+0.000000j  0.000000+0.000000j  0.000000+0.000000j]
 [-0.000000+0.000000j  0.000000+0.000000j  0.000000+0.000000j
   0.000000+0.000000j  0.001525+0.000000j -0.000000+0.000000j]
 [-0.001285+0.000000j  0.010948+0.000000j -0.017277+0.000000j
   0.000000+0.000000j -0.000000+0.000000j  0.032874+0.000000j]]
(0.0737569627134323+0j)
[[ 0.000170-0.017754j  0.000284+0.000003j -0.000441-0.000004j
```

2.B.1 RDMs In addition to the above API higher order density matrices in addition to hole densities can be calculated.

```
rdm2 = lihwfn.expectationValue('i^ j k l^')
if Print:
    print(rdm2)
rdm2 = fqe.expectationValue(lihwfn, 'i^ j^ k l', lihwfn)
if Print:
    print(rdm2)
```

```
[[[[-0.000047+0.000000j -0.000260+0.000000j  0.000445+0.000000j
    -0.000000+0.000000j -0.000000+0.000000j -0.001287+0.000000j]
   [ 0.000308+0.000000j  0.048269+0.000000j -0.073761+0.000000j
    -0.000000+0.000000j -0.000000+0.000000j -0.010927+0.000000j]
   [-0.000437+0.000000j -0.073761+0.000000j  1.987537+0.000000j
    -0.000000+0.000000j -0.000000+0.000000j  0.017274+0.000000j]
   [ 0.000000+0.000000j -0.000000+0.000000j -0.000000+0.000000j
     1.998412+0.000000j -0.000000+0.000000j -0.000000+0.000000j]
   [ 0.000000+0.000000j -0.000000+0.000000j -0.000000+0.000000j
    -0.000000+0.000000j  1.998412+0.000000j  0.000000+0.000000j]
   [ 0.001283+0.000000j -0.010927+0.000000j  0.017274+0.000000j
    -0.000000+0.000000j  0.000000+0.000000j  1.967051+0.000000j]]

  [[-0.000544+0.000000j -0.096416+0.000000j  0.147514+0.000000j
```

2.B.2 Hamiltonian expectations (or any expectation values)

```
li_h_energy = lihwfn.expectationValue(lih_hamiltonian)
if Print:
    print(li_h_energy)
li_h_energy = fqe.expectationValue(lihwfn, lih_hamiltonian, lihwfn)
if Print:
    print(li_h_energy)
```

```
(-8.87771957038547+0j)
(-8.87771957038547+0j)
```

2.B.3 Symmetry operations

```
op = fqe.get_s2_operator()
print(lihwfn.expectationValue(op))
op = fqe.get_sz_operator()
print(lihwfn.expectationValue(op))
op = fqe.get_time_reversal_operator()
print(lihwfn.expectationValue(op))
op = fqe.get_number_operator()
print(lihwfn.expectationValue(op))
```

```
(-1.8969111426479763e-23+0j)
0j
(1.0000000000000018+0j)
(4.000000000000007+0j)
```


Last updated 2023-05-02 UTC.