ED methods with QuSpin

Phil Weinberg

October 7, 2020

Exact diagonalization (ED) and Quantum Many-body Systems.

Many interesting experimental setups are not in a position to probe many-body systems with non-pertubative couplings in order to understand certain phenomena for example:

- Quantum phase transitions.
- Quantum thermalization and Many-body localization.
- strongly out of equilibrium dynamics.

Exact diagonalization methods are limit to small system sizes ($N \sim O(20)$) only by the memory and calculation time requirements.
One tool for these kind of calculations is called QuSpin. An open source python library for doing ED calculations.

Many-Spin Hilbert Space

- Hilbert space for spin-1/2 consists of 2^N states.
- We define kets of spin configurations for N spins with site index increasing from left to right $|\overrightarrow{\uparrow\downarrow\downarrow\ldots}\rangle$.

These spin configs can be mapped to integers using binary numbers:

$$|\uparrow\downarrow\downarrow\ldots\rangle \rightarrow |100\ldots\rangle \rightarrow |2^N - b_s - 1\rangle$$

or...

$$|\uparrow\downarrow\downarrow\ldots\rangle \rightarrow |011\ldots\rangle \rightarrow |b_s\rangle$$

which for $N = 2$:

$$|\uparrow\uparrow\rangle \rightarrow |11\rangle \rightarrow |0\rangle$$
$$|\uparrow\downarrow\rangle \rightarrow |10\rangle \rightarrow |1\rangle$$
$$|\downarrow\uparrow\rangle \rightarrow |01\rangle \rightarrow |2\rangle$$
$$|\downarrow\downarrow\rangle \rightarrow |00\rangle \rightarrow |3\rangle$$

Two-Spin Example

explicitly we can construct some of the operators for two spins. σ_i^α are the pauli-matricies for the i^{th} spin.

$$\sigma_0^z\sigma_1^z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\sigma_0^x = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\sigma_1^x = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

H-space:

$$|\uparrow\uparrow\rangle \to |11\rangle \to |0\rangle$$
$$|\uparrow\downarrow\rangle \to |10\rangle \to |1\rangle$$
$$|\downarrow\uparrow\rangle \to |01\rangle \to |2\rangle$$
$$|\downarrow\downarrow\rangle \to |00\rangle \to |3\rangle$$

example calculation:

$$\sigma_0^x|0\rangle = \sigma_0^x|\uparrow\uparrow\rangle = |\downarrow\uparrow\rangle = |2\rangle$$
$$\sigma_0^x|1\rangle = \sigma_0^x|\uparrow\downarrow\rangle = |\downarrow\downarrow\rangle = |3\rangle$$
$$\sigma_0^x|2\rangle = \sigma_0^x|\downarrow\uparrow\rangle = |\uparrow\uparrow\rangle = |0\rangle$$
$$\sigma_0^x|3\rangle = \sigma_0^x|\downarrow\downarrow\rangle = |\uparrow\downarrow\rangle = |1\rangle$$

Non-Branching Operators

Previous example showed only one matrix element per column. This turns out to be general for any product Pauli-matrices e.g.

$$\left(\prod_j \sigma_{i_j}^{\alpha_j}\right)|s\rangle = M|s'\rangle$$

individual operators only act on the local H-space:

$$\begin{aligned}
\sigma_i^z|...\uparrow...\rangle &= |...\uparrow...\rangle, \quad \sigma_i^z|...\downarrow...\rangle = -|...\downarrow...\rangle \\
\sigma_i^x|...\uparrow...\rangle &= |...\downarrow...\rangle, \quad \sigma_i^x|...\downarrow...\rangle = |...\uparrow...\rangle \\
\sigma_i^y|...\uparrow...\rangle &= i|...\downarrow...\rangle, \quad \sigma_i^y|...\downarrow...\rangle = -i|...\uparrow...\rangle
\end{aligned}$$

These kinds of operators are called *non-branching* operators. Operators also commute in the sense that:

$$\sigma_i^\alpha \sigma_j^\beta |s\rangle = \sigma_j^\beta \sigma_i^\alpha |s\rangle \quad (i \neq j) \tag{1}$$

QuSpin, python package for ED

QuSpin is ED package for python which does lots of different kind of ED calculations:

- Construct both local and non-local operators with both static and time dependent coefficients for Spin, Boson, and Fermion systems.
- Calculate Eigenvalues and Eigenvectors of operators.
- Calculate time evolution under the Schroedinger equation with operators.
- Calculate entanglement entropy and partial traces of quantum states.
- ...

check out the website: http://weinbe58.github.io/QuSpin/ and you can find an in depth introduction: SciPost Phys. 2, 003 (2017).

Installing python+QuSpin(all operating systems):

- install Anaconda Python package manager for your OS.
- type `conda install -c weinbe58 QuSpin` into command line and follow prompts.

Notes about python:

- Python run with an interpreter, to run script from command line type: `python <name_of_script>.py`

Example: Transverse field ising chain. Hamiltonian is given by:

$$H(s) = -s \sum_{i=0}^{L-1} \sigma_i^z \sigma_{i+1}^z - (1-s) \sum_{i=0}^{L-1} \sigma_i^x$$

with periodic boundary conditions $i = L = 0$. Lets go over how to build our hamiltonian for a fixed value of s.

setting up a script, it is good practice to import all neccesary packages before writing any of your own code:

```
1 from quspin.operators import hamiltonian # Hamiltonians and operators
2 from quspin.basis import spin_basis_1d # Hilbert space spin basis
3 import numpy as np # generic math functions
```

Next we set the parameters of our Hamiltonian:

```
5 ##### define model parameters #####
6 L=4 # system size
7 s=0.4
```

Next we define the Hilbert space which we plan on using by defining a basis object. This object is needed to tell QuSpin what kind of many-body operators to construct.

```
8 ##### construct basis
9 basis=spin_basis_1d(L,S="1/2",pauli=True)
10 print(basis) # displays basis states.
```

Next we need to tell QuSpin which operators to construct.

Using QuSpin 3

- In QuSpin, operators $J\sigma_{i_1}^{\alpha_1}\cdots\sigma_{i_n}^{\alpha_n}$ are defined by an n-letter string "$\alpha_1...\alpha_n$" representing operator types, $\alpha_i \in \{+,-,z,x,y\}$ together with a site-coupling list $[J,i_1,...,i_n]$ which holds the lattice site index i for each spin operator in the string.
- For a given operator string we associate with it a list of site-couplings.
- Operator strings are split between static and dynamic pieces where time dependence is given by a user defined function.

Examples:

operator	string	site-coupling list
$-s\sum_{i=0}^{L-1}\sigma_i^z\sigma_{i+1}^z$	`"zz"`	`[[-s,0,1],...,[-s,L-1,0]]`
$-(1-s)\sum_{i=0}^{L-1}\sigma_i^x$	`"x"`	`[[-(1-s),0],...,[-(1-s),L-1]]`

Now in the code we define the site-coupling lists for the ising term and the transverse field, and construct the Hamiltonian object:

```
11 # define PBC site-coupling lists for operators
12 x_field=[[-(1-s),i] for i in range(L)]
13 J_nn=[[-s,i,(i+1)%L] for i in range(L)] # PBC
14 # static and dynamic lists
15 static=[["zz",J_nn],["x",x_field]]
16 dynamic = []
17 ###### construct Hamiltonian
18 H=hamiltonian(static,dynamic,dtype=np.float64,basis=basis)
```

variable `H` will now be used to calculate all sorts of things. In this example we show how to diagonalize the matrix using a few different methods:

```
19 ##### various exact diagonalisation routines #####
20 # calculate entire spectrum only
21 E=H.eigvalsh()
22 # calculate full eigensystem
23 E,V=H.eigh()
24 # calculate ground state
25 Emin,psi_0=H.eigsh(k=1,which="SA",maxiter=1E4)
```

Full code: QuSpin_intro.py

```
1  from quspin.operators import hamiltonian # Hamiltonians and operators
2  from quspin.basis import spin_basis_1d # Hilbert space spin basis
3  import numpy as np # generic math functions
4  #
5  ##### define model parameters #####
6  L=4 # system size
7  s=0.4
8  ##### construct basis
9  basis=spin_basis_1d(L,S="1/2",pauli=True)
10 print(basis) # displays basis states.
11 # define PBC site-coupling lists for operators
12 x_field=[[-(1-s),i] for i in range(L)]
13 J_nn=[[-s,i,(i+1)%L] for i in range(L)] # PBC
14 # static and dynamic lists
15 static=[["zz",J_nn],["x",x_field]]
16 dynamic = []
17 ###### construct Hamiltonian
18 H=hamiltonian(static,dynamic,dtype=np.float64,basis=basis)
19 ##### various exact diagonalisation routines #####
20 # calculate entire spectrum only
21 E=H.eigvalsh()
22 # calculate full eigensystem
23 E,V=H.eigh()
24 # calculate ground state
25 Emin,psi_0=H.eigsh(k=1,which="SA",maxiter=1E4)
```

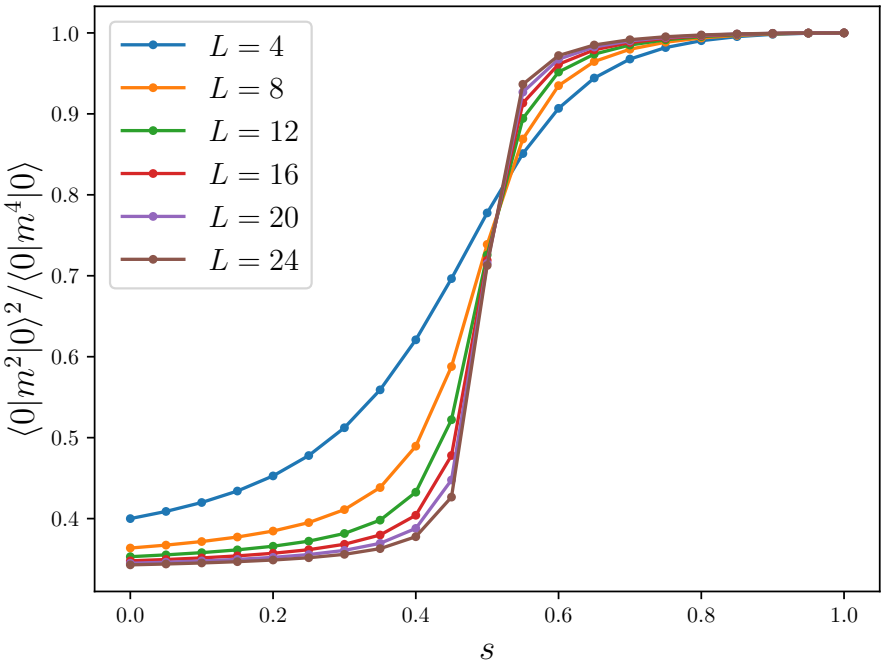
More complicated code, looping over values of s: `QuSpin_qcp.py`

```
1  from quspin.operators import hamiltonian # Hamiltonians and operators
2  from quspin.basis import spin_basis_1d # Hilbert space spin basis
3  import numpy as np # generic math functions
4  import matplotlib.pyplot as plt # plotting results
5  #
6  ##### define model parameters #####
7  L=10 # system size
8  n_points = 21 # number of points to calculate
9  ##### construct basis
10 basis=spin_basis_1d(L,S="1/2",pauli=True)
11 # define PBC site-coupling lists for operators
12 x_field=[[-1,i] for i in range(L)]
13 J_nn=[[-1,i,(i+1)%L] for i in range(L)] # PBC
14 m2_list = [[1.0/L**2,i,j] for i in range(L) for j in range(L)]
15 ###### construct Hamiltonians
16 Hzz=hamiltonian([["zz",J_nn]],[],dtype=np.float64,basis=basis)
17 Hx=hamiltonian([["x",x_field]],[],dtype=np.float64,basis=basis)
18 M2=hamiltonian([["zz",m2_list]],[],dtype=np.float64,basis=basis)
19 ###### calculate Magnetization as a function of s.
20 s_list = np.linspace(0,1,n_points)
21 M2_values = np.zeros_like(s_list) # creates array of zeros shaped like s_list.
22 for i,s in enumerate(s_list): # loop over s-values
23     print("calculating s={}".format(s))
24     E0,psi0 = (s*Hzz+(1-s)*Hx).eigsh(k=1,which="SA")
25     psi0 = psi0.reshape((-1,)) # flatten array.
26     M2_values[i] = M2.expt_value(psi0).real
27 ###### plotting results
28 plt.plot(s_list,M2_values,marker=".")
29 plt.show()
```

Using QuSpin 7

Using lattice symmetries like translational invariance or parity, the system sizes one can calculate increases. In QuSpin one can take advantage of these symmetries by simply specifying them when constructing the basis object:

```
basis=spin_basis_1d(L,S="1/2",Pauli=True,kblock=...)
```



Bonus example, time evolution: `QuSpin_anneal.py`

Using bi-conjugate gradient to solve for spectral functions:

$$S(\omega) = -\frac{1}{\pi}\langle 0|A^\dagger \frac{1}{\omega + i\eta + E_0 - H} A|0\rangle$$

define: $|A\rangle = A|0\rangle$, solve

$$(\omega + i\eta + E_0 - H)|x(\omega)\rangle = |A\rangle \tag{2}$$

then,

$$S(\omega) = -\frac{1}{\pi}\langle A|x(\omega)\rangle \tag{3}$$

Bonus example: `QuSpin_spectral_function.py`